ADAPTATION OF OPEN SOURCE PULSEQ PULSE SEQUENCE WRITING TOOLBOX FOR VARIAN COMPATIBILITY

A Thesis

by

COURTNEY CLARY BAUER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee, Steve Wright
Committee Members, Raffaella Righetti
Laszlo Kish
Jay Griffin
Head of Department, Miroslav Begovic

December 2018

Major Subject: Electrical Engineering

ABSTRACT

The current status of education, within the field of magnetic resonance imaging, is largely dependent on the system being employed. In particular, the pulse sequence programming software is often system specific. Due to this specificity, there is often a shift in focus when it comes to teaching pulse sequence programming: instead of the focus being on the pulse sequence writing itself, it is often focused on the system specific software and languages. While this may not prove to be a particular issue within industry, when it comes to introducing pulse sequence programming to students and potential members of the field, it is less desirable to provide system specific skills rather than those that can be applied across all platform. This thesis seeks to introduce a new interpreter module for the open source pulse sequence programming toolbox, Pulseq.

At Texas A&M Universities Magnetic Resonance Research Laboratory, there are two 4.7T Varian scanners presently employed for both research and education applications. Varian systems are now considered legacy systems as they are no longer actively developed or supported. Given this status, it becomes difficult to justify students continuing to learn pulse sequence programming in the Varian language. By introducing an interpreter module that allows for students to write pulse sequence files using Pulseq, and in turn generate Varian compatible scripts, students can develop skills that are both platform independent, as well as marketable within the MR industry.

DEDICATION

This work is dedicated to my parents and my grandparents who have endlessly supported and encouraged all my pursuits my entire life – you all have consistently reminded me that there are no reasons why I cannot do everything I want to in life.

A special thanks to my dad, who has always gladly been my sounding board. Thank you for your patience, from our stressful calculus tutoring sessions back in high school, to the joys of celebrating academic success as a graduate student. I would not have wanted to do this without you.

A final note that the inspiration for my studies has been driven primarily by my horses – this is for them too. For Novarredo and Tosca, this work serves as a reminder of your impact on my life. For Heino, Picasso, Tira, Rita, and Shilo, here is to many more years of health for you all.

ACKNOWLEDGEMENTS

Of course, I must acknowledge the guidance and support I have received from my committee chair, Dr. Wright, as well as my committee members, Dr. Righetti, Dr. Kish and Dr. Griffin. Throughout the course of my academic career, they've listened to my ideas, views, and goals as I developed my understanding of medical imaging. Their patience with answering my seemingly never-ending questions has been greatly appreciated.

I must also thank my parents and grandparents, without their support none of this would have been possible.

CONTRIBUTORS AND FUNDING SOURCES

This work was supported by a thesis committee consisting of Dr. Steve Wright, Chair, Dr. Raffaella Righetti, and Dr. Lazlo Kish of the Department of Electrical Engineering and Dr. Jay Griffin of the Department of Large Animal Clinical Sciences.

The open source Pulseq toolbox was developed by Layton et al. at University Medical Centre Freiburg, and published online in Magnetic Resonance in Medicine in June 2016. Additionally, it is linked at the ISMRM's MRI Unbound page (https://www.ismrm.org/mri_unbound/sequence.htm). This toolbox is the basis that provides the sequence writing platform which the interpreter module makes Varian compatible. For the sake of being thorough, the toolbox is also published on GitHub (http://pulseq.github.io), with the open-source licensing found at (https://github.com/pulseq/pulseq/blob/master/LICENSE).

All other work conducted for the dissertation was completed by the student independently.

Graduate study was partially supported by a teaching assistantship and a grader position from Texas A&M University.

NOMENCLATURE

MRSL	Magnetic Resonance Systems Laboratory
MR	Magnetic Resonance
RF	Radiofrequency
ISMRM	International Society of Magnetic Resonance in Medicine
T	Tesla

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

1. INTRODUCTION

The present status of pulse sequence programming has a dual focus that does not lend itself well to platform independent skill development. In order to learn how MR images are obtained, knowledge of pulse sequence design and programming is a fundamental, as it allows for students to understand how gradient pulses directly affect images acquired. Associated with pulse sequence programming is timings, gradient strengths, and their effects on the physical dynamics of the source of signal – often referred to as spins [2]. By introducing a tool, such as the open-source Matlab toolbox Pulseq introduced by Layton et al. in 2016 [1]. Texas A&M's Magnetic Resonance Systems Laboratory will not only allow for students to focus on learning how to write pulse sequences, but allow for skill development that can be utilized outside of the lab's legacy Varian scanners due to Pulseq's support of the mainstream Siemens, GE and Bruker scanners [1]. The work presented in this thesis aims to introduce the Pulseq-Varian interpreter module as a means for users to obtain and develop pulse sequence programming skills in a system-independent environment that allows the pulse sequence writing to become the main focus of their efforts, rather than on system specific nuances.

This project is presented with a bit of supplemental information of the student's personal motivation for taking on this project. As long-term career goals involve working in development of low-field MRI technologies, with a particular interest in applications to large animal veterinary diagnostic imaging, a project that allowed for an understanding of system design and control to be built was an appropriate match to the student's aspirations. Given that interfacing the two environments would require not only an understanding of them individually, but also of the MR theory required by the application, the skill set of the student closely aligned with that of the task.

2. BACKGROUND

2.1 Long Term Objectives

2.1.1 Current System Design in Equine MRI

Presently in the field of equine diagnostic imaging, magnetic resonance imaging (MRI) is used primarily in visualization of both soft tissue and bone condition [3]. Often, MRI is used as a means to either identify a lesion within a previously localized area, or as a method to obtain more information about a known lesion [3]. Presently, there are two primary manufacturers that market specifically to the equine industry, Hallmarq and Universal Medical Systems [3].

Universal Medical Systems builds scanners with 0.2 -0.3T field strength, typically requiring the equine patient to be placed under general anesthesia in order to have the region of interest inside isocenter of the magnet [3]. Universal Medical Systems retails and offers several scanner options, ranging from small systems designated for imaging of the distal limb to larger, rotating magnets that can be used for imaging the head, neck, or even stifle [http://www.veterinary-imaging.com/vet-mr.php]. By contrast, Hallmarq's system, Standing Equine MRI, has a comparable field strength at 0.28T, but only requires that the patient be sedated during scanning [4]. The trade-offs between the two will be discussed further in Section 2.1.2.

It is of note that many large animal imaging centers employ high field scanners designed for human use as well, with special considerations taken in order to accommodate a large animal patient. This typically includes the use of specialized, custom made equipment that is both functional in supporting a large animal patient, as well as being MRI compatible. MRI compatible anesthetic equipment, must also be considered, as high field scanners typically require the patient to be placed under general anesthesia.

2.1.2 Need for Advancement in Large Animal MRI Systems

The takeaway is that there are two schools of MR imaging systems for large animal diagnostic imaging: those that require the patient be placed under anesthesia, and those that do not. As there are clear tradeoffs between the two systems, it becomes a focus to decide what the best course of action is for diagnosing a given pathology. The main considerations to be made are with the risks associated with placing a large animal patient under anesthesia, the general costs of operating and the accessibility associated with high-field or large low-field systems, and finally, the tradeoff of image quality associated with a lower field magnet.

One of the primary detriments to most equine MRIs is the need for the patient to be placed under general anesthesia. General anesthesia, even in human medicine, requires careful thought and planning, and even still there are risks to be considered []. These risks are exacerbated when scaled for the large animal patient, especially due to the nature of a prey animal. Placing the equine patient under general anesthesia requires not only careful administration of the anesthesia, but also require planning for the animal to be lowered, placed onto an MRI compatible table, and maneuvered to the magnet room. As the risk for lung collapse become greater the longer the animal is under anesthesia, careful planning of optimal sequences is typically required, often with adjustments made as images are collected [3]. After imaging, the risks are not over, as there is still a chance for the animal to be injured while recovering from anesthesia . Additionally, it must be noted that due to the chances that the animal may have a poor reaction coming out of anesthesia, often cases where fractures are suspected are not considered for MRIs, as the recovery poses a risk for furthering the damage as the animal recovers from anesthesia. Also to be considered are the chances of post-anesthesia colic (PAC). A study claimed that PAC was the most

prominent cause of death in equine patients that had been placed under anesthesia [5]. There are claims that the mortality rate associated with anesthesia is 0.63-0.9% [6]. However, these have been refuted by [1], as the study from [6], focused on the first months of MRI operation at the facility, and mortality rates were observed to decrease as the facility became more experienced with the system and the anesthetic protocols associated with equine imaging. While this is an observation worth noting, it hardly resolves the issue at hand.

In order to use a high-field system, or even a large low-field system, it is necessary that considerations be made from the very beginning – from the design of the magnet room, to the cost of operation and return on investment, to the accessory equipment that needs to be obtained in order to use the system. While some systems come as packaged units meant to function as satellite sites to the operating clinic [4], others require a magnet room to be constructed prior to installation. Magnet rooms often require RF shielding, all of which must also be able to accommodate maneuvering an anesthetized large animal patient [4]. It should also be noted that there are considerations to be made for accessory equipment, as everything from the table the patient is placed on to the anesthesia equipment and tools must be made sure to be MRI compatible as to insure safety.

Finally, there are several factors that directly limit or influence image quality in a standing magnet. Presently, the Hallmarq Standing Equine MRI has a field strength of 0.28T, which directly limits the potential image quality of any scans obtained. This is due to the relationship of signal to noise ratio (SNR) associated with a magnet's static field strength. Additionally, the field created using two pole pieces in a C-shaped magnet typically have smaller maximum field of views than their Halbach counterparts, due to the

small homogenous region located within. Logically, this directly limits the region which can be imaged, especially when regarding image quality.

The main contributor of artifacts in standing scanners is motion. Even in the standing, sedated patient that seems to be still, the patient's breathing alone causes notable motion artifacts to be observed. These effects, coupled with the reduced image quality resulting from the lower field strength, have the potential to make an image unusable. There are currently many methods of motion compensation, most commonly used is specialized pulse sequences that employ navigator echoes [1]. Use of navigator echoes allows for compensation to resolve motion artifacts, but does not necessarily ensure that that the region of interest stays within the homogenous region of the magnet to produce the optimal quality image.

2.1.3 Low Field Magnet Inspiration

Some inspiration to be considered for further development of equine diagnostic targeted scanners is the lightweight portable MRI scanner without gradient coils, developed [7], which is a permanent, low field magnet that organizes magnets in what is referred to as a Halbach array. The Halbach array places permanent magnets in a ring such that the magnets produce a highly homogenous field, have very little stray field, and are relatively cheap compared to their electromagnetic counterparts [8]. The novelty of the low field, gradient magnet, however, is in the encoding scheme. As a result of the novelty of their approach, Cooley et al., have been able to lessen the restrictions on homogeneity by using the inhomogeneities as a means for spatial encoding [7]. As this approach is not supported by traditional reconstruction methods compatible with linear gradient fields, the B_0 field of the magnet must be well mapped within the magnet such that reconstruction is possible [7]. The second low field development of note is the concept of the NMR Cuff,

introduced by [9]. The NMR Cuff proposes a hinged magnet that has the capabilities to be opened and closed without the aid of outside equipment [9]. Interest here would be in the scalability of such a product for applications in diagnostics.

2.1.4 Long Term Goal of System Design

The student's long-term career goals center around improving equine diagnostic imaging. Having lost a horse previously due to an event triggered during recovery from general anesthesia, and had other horses need to be placed under general anesthesia for a variety of procedures, the risks associated are well known to the student. Additionally, drawing on the advancements mentioned above, the ability to make MRI machines more accessible is now within reach, as the low field model presented has a lesser cost of manufacturing, lesser power requirements, and potentially allows for a wider range of applications for its use. Of particular interest of the student is the potential for merging the technologies, looking at novel magnet development, and pursuing low field imaging technique advancements.

2.2 Project Selection

2.2.1 Selecting the Pulseq-Varian Project

The Pulseq – Varian Interpreter Project was a natural fit for the student, as the skills required to develop the interpreter were already possessed or just within reach of the student. Evaluation of Pulseq for interfacing with the Varian system was fundamentally based on compatibility with the Varian system as well as ease of use. Being that the users for the targeted application in the Intro to MRI and MRS course will likely be newly introduced to MR concepts and theory, the Pulseq toolbox needed to be easy to use for those not experienced in pulse sequence programming. Based off the experience as a teaching assistant for the Intro to MRI and MRS course, the student determined that the

toolbox would be easy enough for students to learn, provided clear documentation of the Pulseq functions, and potentially an example. The second evaluation was based in Pulseq's compatibility with the Varian system. The main challenges observed were the discrepancies between gradient coordinates of the environments, timing resolutions, and the capabilities associate with the physical Varian system.

2.2.1.1 Need for a New Pulse Sequence Writing Tool

2.2.1.1.1 Shifting Focus to System Independent Pulse Sequence Writing

Currently in the field of MR research and development, pulse sequence programming is a system dependent process. This leads to pulse sequence programmers needing to become proficient in a number of languages, each specialized to their respective scanners. Introduction of the Pulseq toolbox allows for the focus to be shifted from system specific programming, and instead, allows for the user to focus on the pulse sequence programming outright. Presently, Pulseq has interpreter modules for most mainstream clinical and research magnets, including the Siemens, General Electric (GE), and Bruker systems [1]. As Pulseq is based in the commonly used software Matlab, the learning curve associated with learning the Pulseq environment is considered far less than that of many mainstream MR systems. As mentioned previously, one of the target applications is to provide a pulse sequence environment for the Intro to MRI/MRS course at Texas A&M. Often, this class serves as students' first introduction to MR concepts, and providing a tool that allows them to focus on the pulse sequence programming, and how modification of pulses and parameters modify a sequence. Often, the concepts behind the physical dynamics can prove challenging for some students that are new to MR to grasp through theory alone [2]. Presently, the course has only a single lab where the students get to work specifically with the pulse sequence programming, as they typically utilize the Varian's

built-in macro page for changing pulses and amplitudes indirectly via parameters, and even then, the work is more of a modification than starting from scratch. By pushing students to develop knowledge of how the pulse sequence is written, they in turn build an understanding of the physical dynamics of MR imaging. They must understand how the RF pulse excites a region of interest to a certain angle, how data is written into k-space by the changing gradients, and how different measurements can be obtained via imaging given a few modifications to timing and amplitude.

2.2.1.1.2 Resolves Issue with the MRSL's Legacy Scanner

The Magnetic Resonance Systems Laboratory at Texas A&M presently employs two 4.7T Varian scanners. These scanners are now considered legacy systems, as they are no longer supported or being further developed. The Varian is controlled using compiled C scripts that produce sequences that can then be executed by the scanner. As students join the MRSL's research group, most must learn how to run scans and obtain images or spectrums using the Varian system, which can be a challenge for those that are unfamiliar with the nuances of pulse sequence programming, as there are a variety of functions and macros that the system uses in order to create a pulse sequence [10]. As an alternative, the Pulseq toolbox provides an alternative to having students learn the nuances associated with the legacy system while also allowing them to use a more commonly used environment. At the very least, skills developed with Matlab are more likely to be translated to other applications, as Matlab can be used in a variety of research, academic and even industry related settings.

2.2.1.2 Alignment of Skill Set with the Project

The skill set necessary for this project closely aligned with that possessed by the student. In order to understand, evaluate and integrate the Pulseq toolbox with the existing

Varian system, some programming experience was necessary. As Pulseq is based in Matlab, it was crucial that the individual undertaking the project have experience with data storage and manipulation within the software. Having previously utilized Matlab while a teaching assistant, the student had prior experience with both the storage and manipulation of data, allowing for the design of dynamic storage of sequence parameters based off the determined base pattern. On the other hand, the project also required the ability to work with C-based Varian programming. With prior exposure to C, and proficiency in its object-oriented daughter, C++, learning the Varian flavor of C was a manageable task. That is, rather than having to direct focus on learning the syntactic structure of the generated Varian compatible script, the student was able to direct attention to learning the nuances of how the pulse sequence script controls the scanner.

2.2.1.3 Directly Supports Open Source Initiative in MR Research

The Pulseq environment is directly supported by the ISMRM's open source initiative, MRI Unbound [11]. The goal of the MRI Unbound initiative is to support the open source development of multi-platform compatible tools [11]. A variety of open source software, data sets, and formats are provided there, as a means of creating a more unified initiative and resolving some of the discrepancies within the field. Presently, there are a number of projects that focus towards this initiative, which include, but are not limited to the development of unified standards for data set formatting [12, 13]. Additionally, the ability to create a pulse sequence that has the capability to be reproduced in different scanners by different manufacturers lends to the ability to verify results. Additionally, the introduction of a Varian compatible interpreter allows for Varian systems to be included in discussions of discrepancies between platforms [14, 15].

2.3 Basis of Understanding of Pulseq Environment

2.3.1 Pulseq Toolbox Overview

The Pulseq toolbox is a compilation of Matlab functions and classes made to assist in pulse sequence writing. The toolbox utilizes a variety of functions that allow for pulse parameters, gradient calculations, acquisitions, and delays to be managed. In order to utilize Pulseq, the parameters of the target system must be known. Utilizing a variety of commands, as shown in the sample Pulseq livescripts attached in Appendix D, the user defines system constraints such as ramp time. In an effort to simplify students' interactions with the scanner limitations, an additional function, targetMagnet.m, specifies the system parameters for the student given an input for either '33cm' or '40cm' -- the bore size being the identifier typically used at the MRSL. This allows for students to reference the limitations imposed by the lab to protect the scanners without having to look them up on the scanner themselves. The toolbox contains a number of useful functions, allowing students to quickly define RF sinc pulses using the "mr.makeSincPulse" function, or create a trapezoidal gradients event using the "mr.Trapezoid" function and providing the desired flat time and flat area of the shape – the ramp times for the trapezoid have already been defined in the system parameters previously determined.

2.3.2 Blocks, Events, and Shapes

The structure of a Pulseq generated sequence is broken into three separate pieces. The at the bottom of the structural pyramid are shapes. Shapes are made of compressed lists of normalized sample values that describe a particular pattern to be played out in the sequence[16]. Within the .seq document, each shape has the following attributes listed: Shape ID number, number of uncompressed samples, and the normalized samples

themselves. It should be noted that these are normalized values in order to provide shape information only – amplitudes are later addressed in the next step up, events.

Compression of shape samples is done using run-length compression of the derivative, meaning that the difference between sample values is measured, and any repetitions in this difference after two are compressed. For example, if a derivative value was seen to be 0.1 twenty times in a row, the first two would be listed, but the third through twentieth value would be replaced with an 18 rather than be listed. This is meant to condense the output format of the .seq document listed below, as a large number of samples that are changing at the same rate (ex. a trapezoidal ramp up) would make the file longer than necessary.

Shapes are the smaller building blocks of what make up Events – that is, the RF and gradient events. Shape profiles are normalized values, meaning that they can be referenced by many different Event IDs, and then scaled based off the desired amplitude of a given event referenced. Event IDs can be categorized into five formats: RF pulses, trapezoidal gradients, arbitrary shaped gradients, ADC events, and finally, delays.
RF pulses are counted as an independent event, thus the number of unique RF events present will be the number of RF events (a typical spin echo, for example, would have two RF events: a 90 degree pulse event and a 180 degree pulse event). RF events have the structure containing an event ID, an amplitude in Hz, the magnitude shape ID, the phase shape ID, then the frequency offset in Hz, and the phase offset in radians [16]. Arbitrary gradients have a comparable structure, as they include the event ID, the amplitude in Hz/m, and the shape ID. It should be noted that arbitrary gradient events are only used when the desired gradient shape is not trapezoidal. In the event of a trapezoidal gradient, the rise, flat and fall times are listed, rather than the shape ID [16]. ADC events are defined by the

number of samples to be acquired, the dwell time, the delay between start of the block and first acquisition sampled, as well as frequency and phase offsets [16]. Delay events are defined only by their duration [16].

Finally, the events make up the highest order of structure in a Pulseq generated ".seq" file, blocks. Blocks contain the information of what exactly goes on during a given timeframe in a sequence. Within each block, there are 6 possible events that can be present, a RF pulse, gradient events in the X, Y, and Z directions, ADC/acquisitions events, and delays. It should be noted that all events within a block are assumed to be started simultaneously, and that the duration of the block is determined by the longest duration event contained within. This presents a limitation to be considered when programming pulse sequences with Pulseq, as it is often considered undesirable for events to take place during trapezoidal gradient ramp up/down times [16].

2.3.3 Overview of Pulseq Sequence File Structure

The file generated by Pulseq is in ".seq" file format. This file format begins with version information of the given sequence. Following that, there is an option for user defined values to be included within a "Definitions" section [16]. While an optional parameter, there will be some definitions required for use of the interpreter module. This will be discussed later when discussing the methodology and development of the interpreter module. The next section lists the block events for a given sequence – which is where the individual event IDs are listed for a given block. As stated previously, each block is unique for a combination of events at a given time in the sequence. Following the listing of all blocks in a sequence are the event libraries: the RF library, gradient libraries (trapezoidal and arbitrary), ADC library, and the Delay Library, which contain the relative

event information previously discussed in Section 2.3.2 above. Following the event libraries are the shape libraries in compressed format [16].

2.4 Basis of Understanding for Varian Environment

2.4.1 Overview of Varian Pulse Sequence Structure

Fundamentally, the structure of a Varian pulse sequence document closely follows that of a common script written in the C language, however there is a learning curve associated with the environment. The Varian programming language provides a diverse set of specialized functions, macros and nuances that must be addressed, making the language difficult to grasp for individuals new to the field of MR imaging. First are the #includes and #define statements, the declaration of static variables, and then the main function. The Varian sequence can typically be divided into two separate structural categories: the first being file declarations, the secondary being the pulse sequence commands themselves, contained within the Varian "pulsesequence()" function [17]. It is important and of note that these sequences must be compiled on the scanner prior to being run, thus declaration of any pulse sequence parameter values must happen inside pulse sequence class.

The first structural element of the Varian pulse sequence file is the standard file declarations. This portion of the pulse sequence file is usually short, as it is where preprocessor directives are used, and typically, static variable declarations are made. All Varian pulse sequence files must include the header file 'standard.h' using the "#include" directive. This header file contains the different classes, commands and macros that allow for use with the Varian system. Additionally, minimum instrument delays are introduced using the "#define" directive. In the Varian sequence Ssems.c, phase tables are also declared as static values [18].

The second structural element is the pulse sequence itself. This part of the script always begins with the "pulsesequence()" function, which as mentioned previously, relates and functions to the Varian system in a manner similar to that of "main()" in a standard C script. Contained within the "pulsesequence()" function are the sequence's variable declarations, system checks, calculation of parameters, as well as the construction of the pulse sequence loop itself. Declared variable values can then be modified via command line without recompiling the sequence each run. Formatting of declaration of variables parallels the declarations followed in a standard C script, as they are based off the type of variable being used – integer, double, etc. This is also where Varian specific functions such as "getval()" are used, as they allow for user defined parameter values to be called [19]. Once the declarations have been completed and all user defined values have been called, the Varian scripts typically follow with calculation of gradient parameters, which are then followed by system checks against limitations in order to ensure safety and prevent damage to the system.

Once the parameters have been defined, the pulse sequence itself can be built. Imaging sequences utilize at least some form of looping the means of encoding data within k-space. The most commonly achieved, for the targeted applications within the introductory course, using phase encoding [20]. Varian has a looping function specifically designed to aid in phase encoding, "peloop()" as it updates and compares to real time variables as a counter method [19]. This command signals the beginning of the pulse sequence base. It is of note that this only causes the sequence to repeat, and does not control the phase encoding gradient [19]. For the interest of this project, there are four types of Varian events that are of interest recognized: RF pulse events, gradient events, ADC/acquisition events, and delays.

RF pulse commands are a mostly standard process on the Varian system, requiring two lines for an RF pulse event to be defined. These first of these two lines sets the RF power levels, and the second plays out the RF pulse. During the execution of an RF pulse using this command, until the RF pulse has been finished being executed, no other lines of the script can be executed [19]. This leads to challenges in the uncommon event where trapezoidal gradients ramp down part way through and RF pulse, explained further below.

Varian gradient commands are the most complicated of the set, as their code requirements are highly dependent on two separate factors. First, they fundamentally vary based on shape – that is, arbitrary gradients use a single command for execution, whereas trapezoidal gradients have different requirements are traditionally "ramped up", held, then "ramped down" as a way to save memory [19]. Following the shape factor is duration – depending on the duration of the gradient event compared to other events being executed at that same time, the command order varies accordingly, and delays may or may not need to be added to properly define gradient durations.

The third event of interest is the ADC/acquisition event. The acquisition event statically requires three commands: activation of a spare line for high speed acquisition, call for the actual acquisition, and deactivation of the spare line [18]. The final event of interest, delays, require a single line of code and function as a typical delay does, where there are no changes present in the sequence [19]. Delays have a variety of applications as they can be used to wait between pulses, or used to maintain a trapezoidal gradient event duration in the event that no other event s are executed within that time [17].

2.4.2 Specialized Functions and Macros

One of the main concerns when creating this interpreter module was being sure to avoid the many macros built into the scanner, often easily mistakable for variable names.

As a means to avoid these in the initial developments, rather than utilize variables, many parameter values are hard coded, with plans for future developments to switch to variables. Due to the complexities of using highly specialized functions, the selection process is boiled down below, with the selected functions explained in detail.

2.4.3 Selection of Applicable Functions

When choosing functions, flexibility was the top priority – the target here was to create an interpreter that allowed for both innovation and ease of use. When looking at specific functions, the ability to customize waveforms and pulses, the inclusion of oblique imaging capabilities, etc., were considered. As a result, careful consideration was taken into consideration when choosing sequence functions.

2.4.3.1 Selection of Applicable Functions: RF Pulses

As mentioned in Section 2.4.1, RF pulses commands have been chosen to constantly require two commands. The first of these two commands, "obspower()" simply sets the power level of the RF pulse based off the value input, which is a calibrated value in dB. The second command has a bit more detail, as it has many more inputs due to its function as executing RF pulses.

The "shapedpulse()" function accepts five inputs. The first input is the shape profile. The shape profile, contained within quotation (as C sees it as a string), is the filename of the shape file contained within the shape library. The shape library must contain that filename with the file extension being ".RF" in order to the shape to be recognized. The second parameter is straightforward: it is the RF pulse duration, in seconds, and the third parameter is the RF phase variable, used for phase cycling. For the purpose of this applications, phase cycling is not determined on a sequence basis, and is

instead kept as a uniform method between files generated. The fourth and fifth parameters are the gating pre- and post - delay for the RF pulse in seconds, respectively.

2.4.3.2 Selection of Applicable Functions: Gradient Pulses

In order to best facilitate the varying cases of gradient control, selecting a gradient command that allows for both oblique slice definition, so that slices can be oriented using Euler angles, and the ability to choose whether or not to complete the gradient command execution before continuing to the next event will be required [19, 20].

The selected function for use was "obl_shapedgradient()" which receives nine inputs. The first three inputs are for the gradient shape profiles in the readout, phase encode, and slice select directions, respectively. The fourth parameter is the duration of the gradient event, followed by the gradient amplitudes in order of readout, phase encoding and slice select. The units on these amplitudes are G/cm. The eight input is an integer that defines the number of times the event is looped. The last input parameter is a "WAIT" or "NOWAIT". When designated as WAIT, the system waits until the command is executed before continuing, and as expected, when designated as NOWAIT, it does not wait for the continuation. Presently, the system does not support dynamic multiple oblique gradient events, due to the interdependency of oblique gradient planes. However, as the MRSL is a systems lab, and that functionality was not initially clearly defined, the original software utilized the WAIT/NOWAIT parameter as a means for having simultaneous events. This leads to a much more flexible and programming environment – though currently not implementable. The beta version of this software is still available in the event the hardware can be found to support these events, it can be updated and implemented. This will be discussed in more detail later. It should be noted that trapezoidal gradients are

defined with calls of this function, one with “ramp_hold” and one with “ramp_down”. The flat time is defined either by a delay or a simultaneous event (ADC or RF pulse) [19].

2.4.3.3 Selection of Applicable Functions: ADC Events and Delays

Due to the simplicity of ADC events and delays, they have been lumped together in this section. An ADC event uses three functions: “sp2on()” and “sp2off()” turn on or off, respectively, a high speed spare line for use during acquisition, while the “acquire()” command takes place between them. The “acquire()” command takes two inputs, the number of points acquired in double format, and the dwell time in seconds, though it is commonly implemented as the inversion of the spectral width [19].

Delay events have a single line of commands, with the command being called, naturally, “delay()”. The delay command takes a single input of the delay duration in seconds [19].

3. METHODOLOGY

3.1 Goals of the Interpreter Module

With the functions to be used selected, the focus turns to creation of the interpreter module. Two main goals are recognized for this project: first, the end product must be both flexible and informative enough to support innovative pulse sequence design, as well as allow students new to MR to make mistakes; and second, it must still be easy to use. In order to meet the goals of flexibility, the interpreter has been designed to support unlikely cases, such as near-simultaneous RF pulses in combination with ADC events. As the Varian does not easily, if at all, support simultaneous transmit and receive, and most of the target users will not utilize that case intentionally, it is currently programmed such that the acquisition immediately follows the completion of the RF pulse. In order to be deemed useful to the undergraduate course, the interpreter needs to be able to produce Varian scripts that are either ready for use, or are requiring minimum modifications. In order to meet the goal of ease of use, the interpreter module is a used single function with two inputs: one for the ".seq" input file name, and a second string that defines the name desired for the output, Varian compatible script.

In order to provide a visual explanation of how the interpreter is implemented in the pulse sequence writing process, the Figure 1 provides a flow chart. The user, on their own device, will be able to download and install the open source Pulseq toolbox, utilizing this toolbox, they will write the pulse sequence and generate the .seq output file. That output file is then input into the interpreter, generating the Varian compatible pulse sequence script. That script is then transferred to the scanner computer, compiled and run, resulting in an image being obtained.

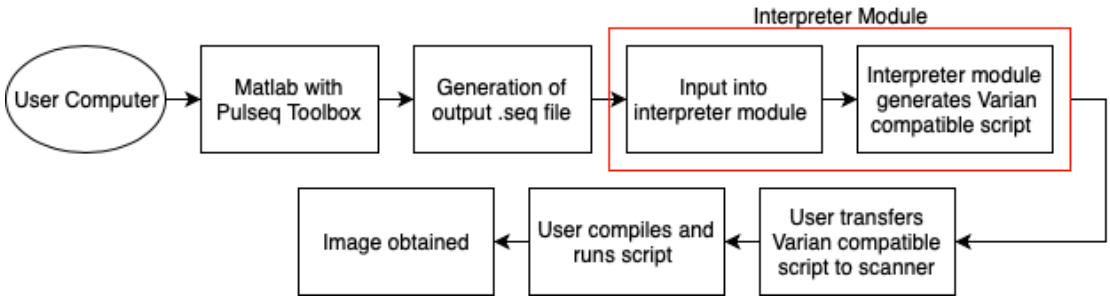


Figure 1: Flowchart displaying where the interpreter module fits within the process

3.2 Bridging the Gap Between the Pulseq Toolbox

3.2.1 Timing Resolution

When taking on this project, the main discrepancy identified was that of the timing resolution. Using Pulseq, every block of the sequence is defined by that of the longest duration event contained within the block. This poses a particular challenge, as it limits what pulse sequences can be written. As a result, for RF and ADC events coinciding with gradients, comparisons have been made and delays added as necessary in order to achieve desired durations, however it should be noted that all events in a block are assumed to begin simultaneously, with the exception of the presence of a trapezoidal gradient. In the event that a trapezoidal gradient is present, no events occur during ramp up or ramp down periods.

3.2.2 Gradient Direction

The second discrepancy recognized was the definition of gradient directions. Pulseq defines gradients directions in terms of G_X, G_Y, and G_Z, and Varian defines gradient direction in terms of G_{RO}, G_{PE}, and G_{SS}. The introduction of obliques resolved this issue, as slice orientation then becomes more flexible and can be set via the Euler angles, and directions were designated. G_X is defined as G_{RO}, G_Y is defined as G_{PE}, and G_Z is defined as G_{SS}.

3.3 Interpreter Development

3.3.1 Recognizing the Basic Sequence

The first step in creating to the interpreter was the organization of relevant parameters. Rather than "reinvent the wheel", the student opted to utilize the "mr.read" function provided within the Pulseq toolbox, as this provided the information in a more directly referenceable format. Once the .seq file was read into Matlab and the data organized within a "struct" object, the basic sequence could be recognized.

In order to recognize the base sequence, a counter is established to record the number of blocks present before the exit conditions are met for a while loop. For the purposes of this interpreter and its target application within the introductory course, the exit conditions were defined as repeated identical RF events following at least one acquisition. It is acknowledged that this is a simplified case that may not hold up when working with more advanced sequences, but has been made easily expandable should it need to be modified.

As a means to count the number of blocks before the exit conditions are met, the interpreter looks for events within each block, starting with a delay. It is assumed that in delay event case, no other events take place, so the delay duration is recorded and the counter is incremented by one. In the event no delay is recorded, the interpreter looks for an RF pulse, if none is present, it looks for gradient events, then for ADC events, recording the necessary parameters into new struct objects for use in the sequence. Once the first ADC event is recorded, it sets a variable that is used as a flag, saying that at least one acquisition has taken place. If no RF pulse have been previously recorded, or the RF pulse is not identical to any previously recorded RF pulses, it is stored, and the follows the same path as if there was no RF event, looking for gradient and ADC events and recording them.

If the RF pulse is repeated, and the ADC event flag has been set to true, then the loop is exited and the block counter is decreased by one – if the ADC event flag is false, then the info is once again stored and other events in that block are recorded. This is more clearly illustrated in the flowchart, shown in Figure 2.

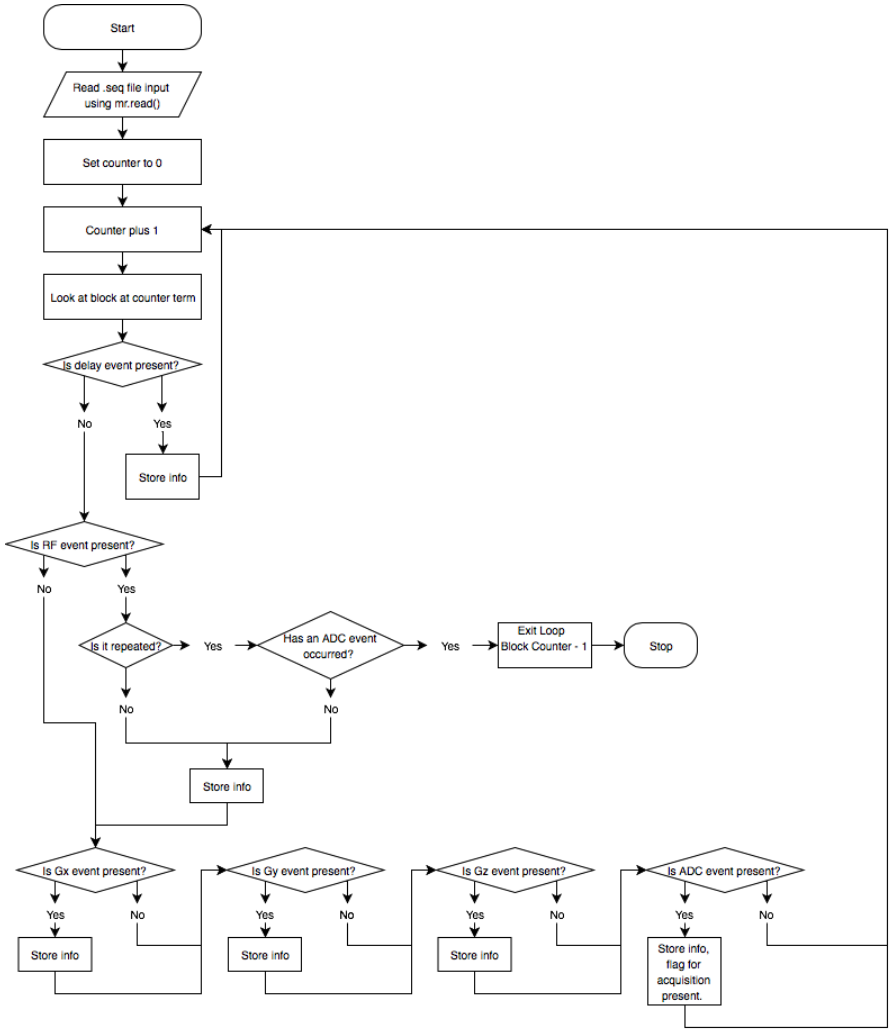


Figure 2: Flowchart Demonstrating the Base Sequence Identification Algorithm

3.3.2 Writing Block Events to File

Once the looping sequence has been structured, the output file can be written. Looping through the saved information from the base sequence identification algorithm for

the number of blocks determined, correctly formatted Varian functions can be written appropriately.

Again, because the delay case allows for the assumption that no other events are present in the block it is looked at first. If there is a delay case, the delay function is written to the output file with the desired duration, and the loop counter is incremented. If no delay is present, then the RF and ADC events are looked at next. If an RF event is in the block the power setting is written first, and a flag is set to true to state that there is an RF event present. If no RF event is present, the flag is set to false. Following the RF event, it is determined whether or not an ADC event is present in the block, and a flag is set again, similar to that of the RF event flag. Next, the number of gradients must be addressed. The appropriate subfunction is determined by the number of gradients active within the block, whether it be one, two or three gradient events. By creating various functions, it allows for different combinations of channels to use the same function. These functions can become rather complex, so only the basic flow of the functions highlighted in Figure 3, below.

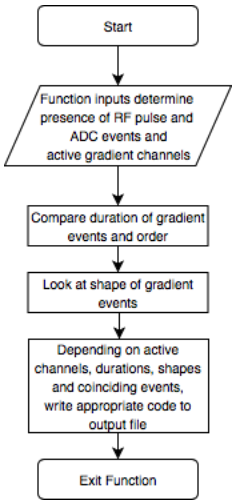


Figure 3: Basic structure of the gradient event output writing subfunctions

As mentioned before in Section 2.4.3.2, there were two different developments of the Gradient Event Output Writing Subfunctions, but the basic structures are identical. In the first version, the one that supported multiple dynamic oblique gradients, duration comparisons were made in order to best format the output sequence. In this version, for example, a trapezoidal gradient could ramp down while the others were maintained. Due to the interdependence of gradient levels on oblique planes, while this code seems to stand in theory, it is not applicable in practice due to the interdependence of oblique gradient planes with each other. After much discussion, as a means to resolve this issue, the second version, included in Appendix A, the gradient comparison seeks to find the longest duration gradient event and set all gradient events in that block to that duration. It should be noted that the scripts included in this thesis will be a static version of the software – updates will continue to be introduced and made available. Contact Texas A&M's MRSL for the most recent version.

In the event that no gradients are present within the block of interest, the RF and ADC flags are assessed, and the proper code is written to the output file without the use of a subfunction. In the event that both flags are set to false, an error is thrown and the script is aborted, as those cases should not exist. A flowchart demonstrating the overall script writing algorithm is shown in Figure 4, below.

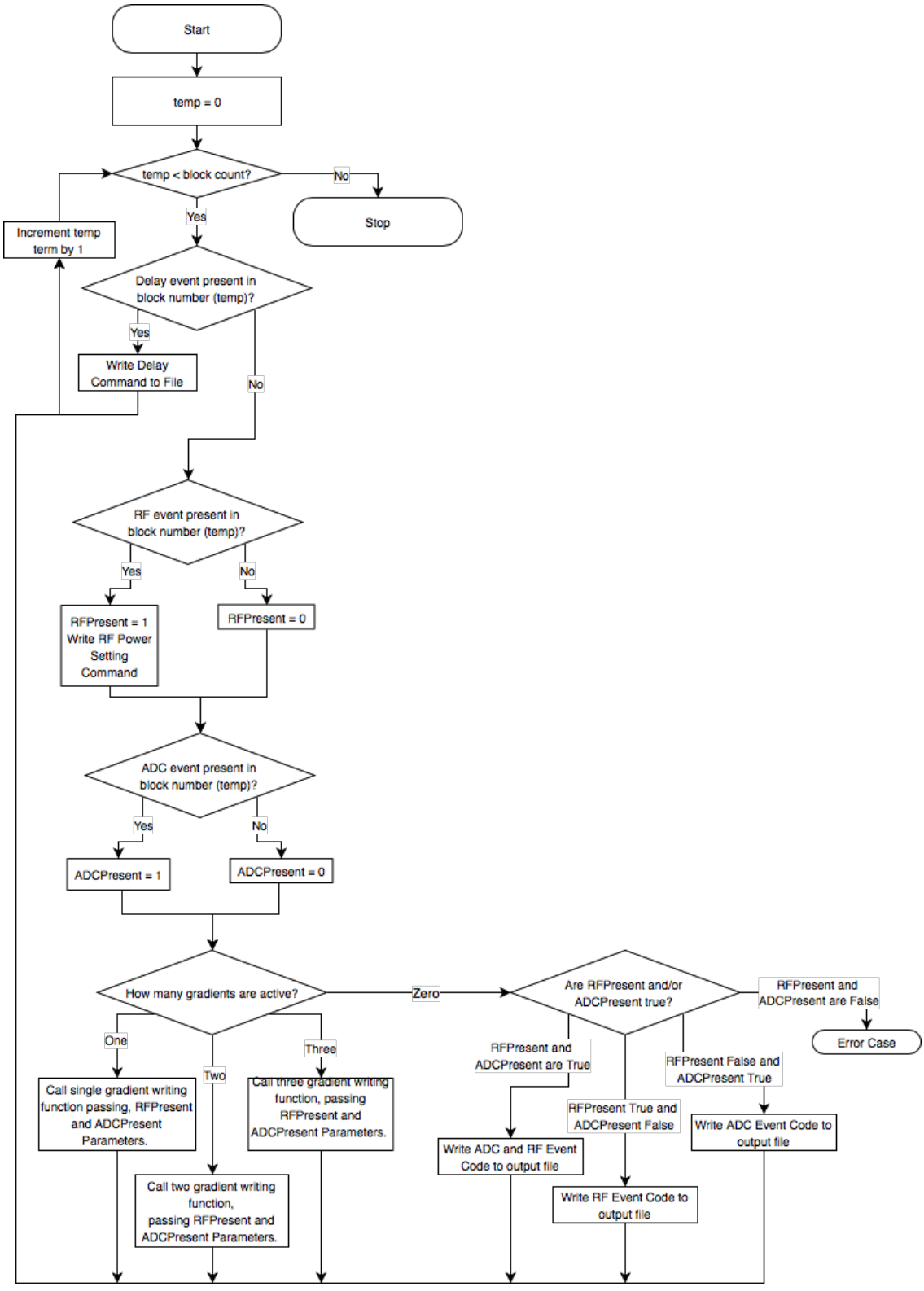


Figure 4: Overall script writing algorithm

3.4 Overview of Documentation

With the integration of this project into use at the MRSL, there are two types of documents to be provided, a user manual for pulse sequence writing using Pulseq at the MRSL, and Matlab Livescript aids, as a means to help TAs for the Intro to MRI/MRS course learn Pulseq while also being able to help students at the same time.

3.4.1 User Manual

The user manual seeks to provide basics in pulse sequence programming as well as MRSL specific examples, as this will likely be used primarily by students in the MRI introductory course. As mentioned previously, in this course, students have the opportunity to run and obtain a variety of images using one of the MRSL's 4.7T scanners. Through the introduction of pulse sequence programming, another layer of understanding can be achieved, as many students attempt to glaze over this fundamental, seeing the concepts as a set of equations rather than a key to imaging as a whole. By encouraging students to develop pulse sequence programming skill, they are required to take a step back, look at the pulse sequence and its relation to the physical dynamics of the system, stronger foundations for MR education can be built. This document seeks to provide function input parameters, units and serve as an MRSL specific document for implementation of Pulseq. The current copy of the user manual is attached in Appendix C.

3.4.2 Livescript Aids

Presently, Matlab Livescript aids are being developed as an aid for integration of the Pulseq environment into the Intro to MRI and MRS course at Texas A&M. Based off the traditional labs, these documents are meant to be provided to teaching assistants and instructors as a way to smoothly integrate the software while simultaneously being able to

aid students as well. Attached in Appendix D are images of the first livescripts developed for Lab 2: Introduction to MR Imaging, and Lab 3: Slice Selection and Frequency Encoding. Matlab Livescripts were selected as the platform for educational aids as the allow for easy display of simultaneous sections that can be modified and viewed side by side.

4. IMPLEMENTATION

4.1 Pulse Sequences and Resulting Images

As a means to evaluate the interpreter's performance, the pulse sequence generated by it needed to be compared against a standard. In order to achieve this, it was decided that a sequence generated by the interpreter module would be compared against an existing Varian-written pulse sequence. A basic spin echo sequence using half-sinusoidal rephase, dephase and phase encoding gradients, Ssems, was selected. Using Pulseq and the interpreter module, a sequence was written to match the Ssems Sequence. The Varian's Ssems, and the interpreter module's spin echo script are attached in Appendices E and F, respectively.

Shown below in Figures 5 and 6 are the screen captures of the scoped outputs from the system during the spin echo sequences. In Figure 5, we see the Varian Spin Echo, containing the half sinusoidal shapes. Figure 6 shows the interpreter generated spin echo sequence. It should be noted here that the introduction of the spoiler gradient during the 180 refocusing pulse was a manual change necessary as a means to facilitate a direct comparison between sequences.

The images were obtained using a large birdcage coil, where the RF pulse power was calibrated independently using the scanner. The phantom contained within is a cylinder, 7.75 cm in diameter, filled with a solution of copper sulfate and containing two ramps within it. The phantom has been positioned such that the ramps are outside the selected 3 mm slice. The parameters used in imaging are shown below, in Table 1.

Repetition Time:	1000 ms
Echo Time:	30 ms
RF Pulse Duration:	2 ms
Phase Encode/Dephase/Rephase Duration:	2 ms
Ramp times:	0.5 ms
RF 90 Power:	37 dB
RF 180 Power:	43 dB
Receiver Gain:	14 dB

Table 1: Image parameter values

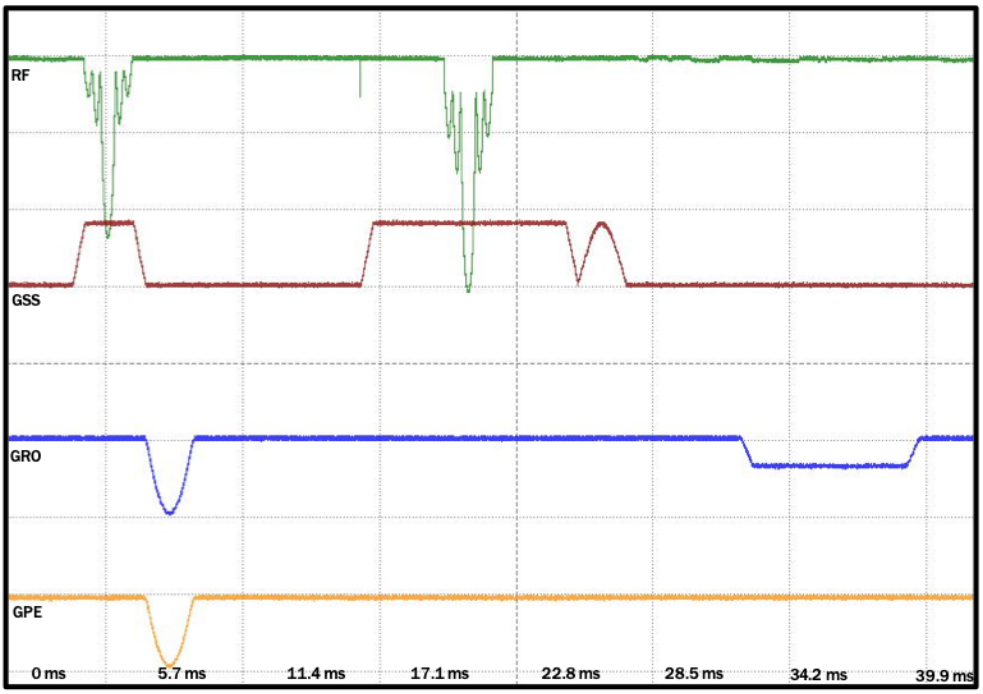


Figure 5: Scoped Varian spin echo

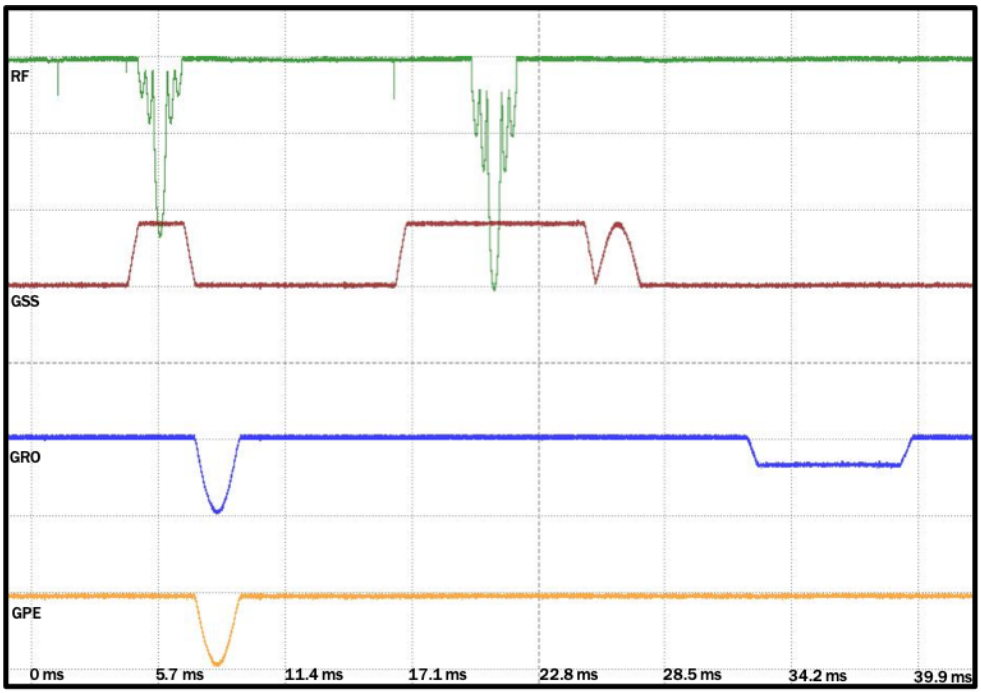


Figure 6: Scoped interpreter generated script spin echo

In the figures containing the scoped output, it should be noted that the green line represents the RF line, the red line is the slice select gradient, the blue line is the frequency encoding gradient, and the yellow line is the phase encoding gradient. The figures above show a near identical scoped outputs, though it should be noted they are not completely identical. For the purposes of comparing the two figures, they are functionally, identical – the only observed difference between the two is the noise on the RF line.

The next task was to compare images. In Figure 7 we have the image resulting from the Varian Ssems script, whereas Figure 8 shows the interpreter generated script image:

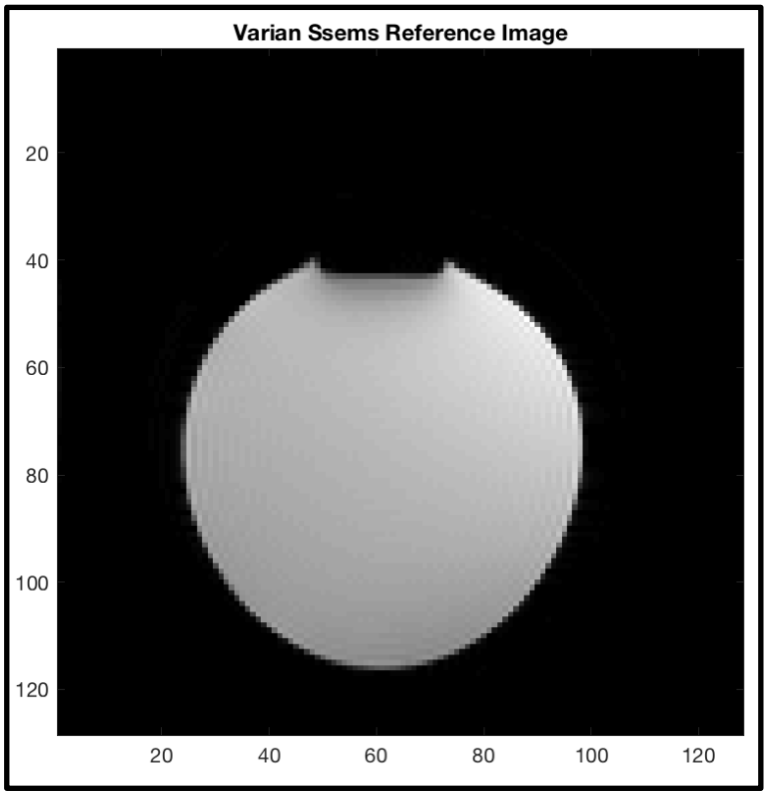


Figure 7: Varian Ssems image

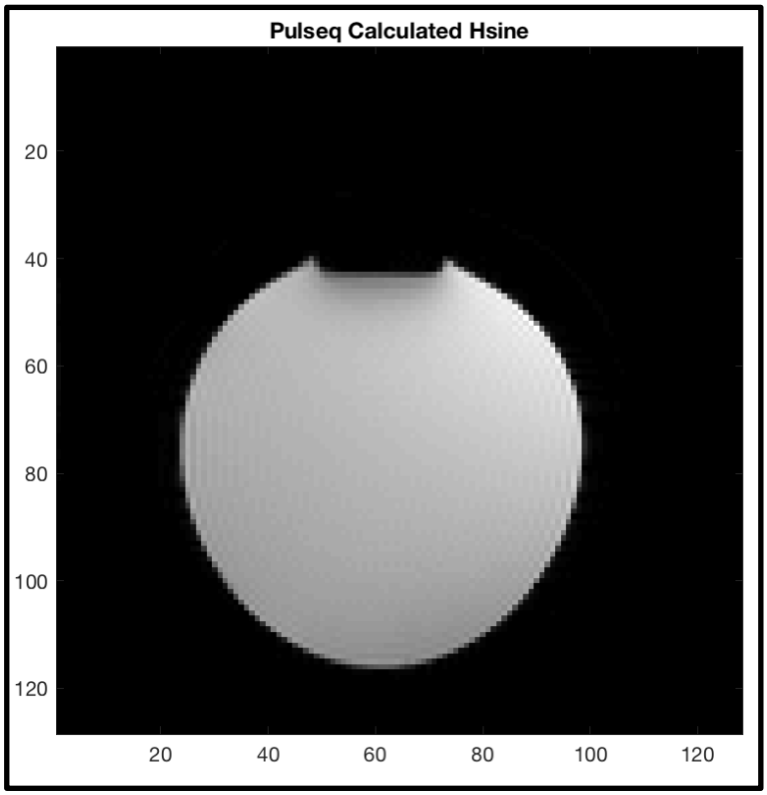


Figure 8: Interpreter generated script image

In comparing the two images, they are observed to be nearly identical, however quantitative measurements can be made using the SNR values. Figures 9 and 10 below show the sample regions for signal, in red, and noise, in blue.

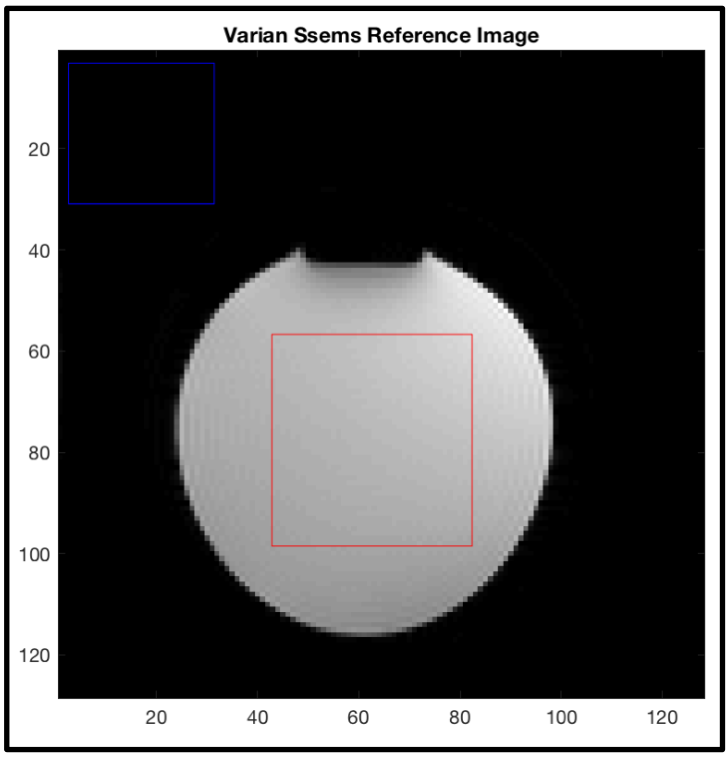


Figure 9: Varian Ssems image with sample regions

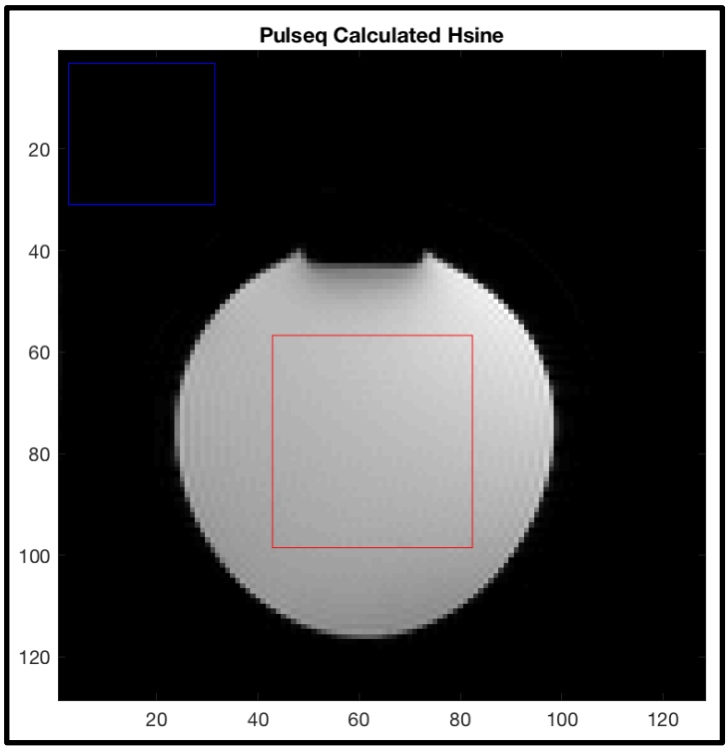


Figure 10: Interpreter generated script image with sample regions

Utilizing the simplest form of SNR, the ratio of average signal over a given region of the sample to the noise in the noise region of the sample, the SNR for these two images are both nearly 26.7 dB. The Varian had a signal value of 6.506, and a noise amplitude value of 0.014, leading to a 26.717dB. The script generated by the interpreter module had a max signal value of 6.496 and a signal value of 0.014, resulting in a SNR of 26.691 dB. As can be seen from the images and data presented here it, it is clear that the two different scripts produce nearly identical images, thus validating the interpreter module's ability to generate usable pulse sequences.

4.2 Further Developments

4.2.1 Introduction of Variables

In future developments of the Pulseq-Varian interpreter module, the ability to dynamically declare variables would be greatly beneficial, as the current version directly programs gradient values as fixed numbers. The introduction of variables would allow for gradient parameters to be modified at the system without generating, transferring, and compiling a new sequence. That being said, this is no small task, as it requires the ability to detect the number of variables needed, name them in a logical and efficient manner, declare values and be able to modify them as needed. While this may be a relatively straightforward task for integrating the ability to change pulse amplitudes, introducing timing variables would require creating algorithms that carefully determine and write equations for timing dependencies and calculations, and is a much more daunting task as the dependencies get more and more complicated, and while ideal for future developments, was beyond the scope of this initial version.

4.2.2 Further Compatibility/Flexibility

Future improvements to make, as Pulseq continues being integrated into employment at the MRSL is the generation of macros as a means of pulse parameter modification. Presently, the interpreter generated script relies heavily on command line controls, which can be daunting to a student before they are comfortable interacting with a machine such at the MR magnet.

Additionally, added functions and flexibility allow for more rapid development of innovative pulse sequences. Introduction of new functions is a straightforward process, taking very little time. Presently, three functions are already working or in progress, those are attached in Appendix B. These, in total, took no more than a couple of hours to write – which is proof in itself in that modifications and new tools can be rapidly developed to allow expansion.

5. CONCLUSION

The Pulseq-Varian Interpreter module introduced in this work has been primarily created as a means to aid in the education in the field of MR applications. By incorporating this tool into the Intro to MRI and MRS class at Texas A&M, students will be able to focus on how to program a pulse sequence, but also be able to build an understanding for the physical dynamics that make MR imaging possible. Additionally, from an educational standpoint, this work allows students to develop platform independent skills that they can take into a career in the MR industry. From a research perspective, this interpreter module has the ability to provide a platform that allows for a researcher unfamiliar with the Varian functions and nuances to rapidly develop a pulse sequence on a Varian system without needing to take the time to learn the smaller system specific details.

Overall, this paper sought to introduce and evaluate the interpreter module created for implementing the open-source Pulseq, and making it communicate with the Varian system. From the results, it is clear that the interpreter can hold its own against the current standard.

REFERENCES

[1] K. Layton, Kroboth, S., Jia,F. , Littin, S. , Yu, H., Leupold,J., Nielsen, J. , Stöcker, T. and M. Zaitsev, "Pulseq: A rapid and hardware-independent pulse sequence prototyping framework," *Magnetic Resonance in Medicine,* vol. 77, no. 4, pp. 1544-1552, 2016.

[2] M. Levitt, *Spin Dynamics*, 2 ed. Chichester: Wiley, 2015.

[3] R. C. Murray, *Equine MRI*. Chichester, Est sussex, UK: Wiley-Blackwell, 2011.

[4] Hallmarq, "Standing Equine MRI," ed. https://indd.adobe.com/view/dcde7039-3193-45f8-a345-60c63767e19d: Hallmarq, 2018.

[5] R. C. Jago, F. Corletto, and I. M. Wright, "Peri-anaesthetic complications in an equine referral hospital: Risk factors for post anaesthetic colic," *Equine Vet J,* vol. 47, no. 6, pp. 635-40, Nov 2015.

[6] P. Franci, E. A. Leece, and J. C. Brearley, "Post anaesthetic myopathy/neuropathy in horses undergoing magnetic resonance imaging compared to horses undergoing surgery," *Equine Veterinary Journal,* vol. 38, no. 6, pp. 497-501, 2006.

[7] C. Z. Cooley *et al.*, "Two-dimensional imaging in a lightweight portable MRI scanner without gradient coils," *Magn Reson Med,* vol. 73, no. 2, pp. 872-83, Feb 2015.

[8] H. Raich and P. Blümler, "Design and construction of a dipolar Halbach array with a homogeneous field from identical bar magnets: NMR Mandhalas," *Concepts in Magnetic Resonance Part B: Magnetic Resonance Engineering,* vol. 23B, no. 1, pp. 16-25, 2004.

[9] C. W. Windt, H. Soltner, D. van Dusschoten, and P. Blumler, "A portable Halbach magnet that can be opened and closed without force: the NMR-CUFF," *J Magn Reson,* vol. 208, no. 1, pp. 27-33, Jan 2011.

[10] *User Guide: Imaging*. Palo Alto, CA: Varian, Inc. , 1999.

[11] (2018, 21- Sep- 2017). *MRI UNBOUND* [Online]. Available: https://www.ismrm.org/mri_unbound/

[12] K. Gorgolewski, Auer, T., Calhoun, V., Craddock, R., Das, S., Duff, E., Flandin, G., Ghosh, S., Glatard, T., Halchenko, Y., Handwerker, D., Hanke, M., Li, X., Michael, Z., Maument, C., Nichols, B., Nichols, T., Pellman, J., Poline, J., Rokem, A., Schaefer, G., Sochat, V., Triplett, W., Turner, J., Varoquaux, G., Poldrack, R., "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments," *Scientific Data,* vol. 3, 2016.

[13] S. Inati, Naegele, J., Zwart, N., Roopchansingh, V., Lizak, M., Hansen, D., Liu, C., Atkinson, D., Kellman, P., Kozerke, S., Xue, H., Campbell-Washburn, A., Sørensen, T., Hansen, M., " ISMRM Raw data format: A proposed standard for MRI raw datasets," *Magnetic Resonance in Medicine,* vol. 77, no. 1, pp. 411-421, 2016.

[14] X. Han, Jovicich, J., Salat, D., van der Kouwe, A., Quinn, B., Czanner, S., Busa, E., Pacheco, J., Albert, M., Killiany, R., Maguire, P., Rosas, D., Makris, N., Dale, A., Dickerson, B., Fischl, B. , "Reliability of MRI-derived measurements of human cerebral cortical thickness: The effects of field strength, scanner upgrade and manufacturer," *NeuroImage,* vol. 32, no. 1, pp. 180 - 194, 2006.

[15] M. Jaring, Krikhaar, R., Bosch, J., "Representing variability in a family of MRI scanners," *Software: Practice and Experience,* vol. 34, no. 1, pp. 69-100, 2003.

[16] K. Layton, Zaitsev, M. , " Open file format for MR Pulse Sequences," ed. University Medical Centre Freiburg, 2015.

[17] *VNMR Pulse Sequences*. Palo Alto, CA: Varian, Inc., 2000.

[18] *VNMR User Programming*. Palo Alto, CA: Varian, Inc., 2000.

[19] *VNMR Command and Parameter Reference*. Palo Alto, CA: Varian, Inc., 2000.

[20] M. K. Bernstein, K. Zhou, X. , *Handbook of MRI Pulse Sequences*. Amsterdam: Elsevier Academic Press, 2004, p. 1040.

Supplemental Sources

These sources were not cited in text; however, general knowledge and understanding were obtained through use of the software, and reading in the manuals.

Pulseq Open-Source Software:

K. Layton, S. Kroboth, F. Jia, S. Littin, H. Yu, J. Leupold, J. Nielsen, T. Stöcker and M. Zaitsev, Pulseq Toolbox. [Open Source, via https://github.com/pulseq/pulseq]. Freiburg, Germany: University Medical Centre Freiburg, 2015.

Varian Manuals:
User Guide: Imaging, Palo Alto, CA: Varian, Inc., 1999.
Getting Started, Palo Alto, CA: Varian, Inc., 2000.

APPENDIX A

Pulseq-Varian Interpreter Software

Note: Provided below is a static copy of the interpreter function that reduces the flexibility of the software. As discussed above, this version sought to make the necessary modifications to avoided the unwanted distortion of coinciding oblique gradient during a change on one gradient channel. This version resolves the issue by analyzing and setting all gradients within a block to that of the longest duration contained within the block. For the most recent copy, please contact the author.

VarianBuild.m

```
%% Commented out Function Declaration of Testing
function [] = VarianBuild_sameDurationGradMod(obj, filename)
%% Check for Existing File With Name
% Check to be sure that the file name is available
loc = pwd;
if exist(fullfile(loc,filename), 'file') == 2  % if file already exists
   fileExists = sprintf('Warning: Existing File Name in Directory:\n%s', filename);
   rename = sprintf('Please Rename File');
   uiwait(warndlg({fileExists,rename} ,'Warning: Existing File Name'));
   loc2 = fullfile(loc,'*.c');
   [realFilename, ~] = uiputfile(loc2,'Save as...');
   fopen(realFilename, 'w');
   realFilepath = fullfile(loc,realFilename);

else % if file does not exist yet
    fopen(filename,'w');
    realFilename = filename;
    realFilepath = fullfile(loc,realFilename);
end
%% Read in the .seq File
nseq = readSeqFile(obj); % Read in .seq file
% Get System defined values:
system_definitions = struct('targSys', nseq.getDefinition('FieldStrength'),...
                    'Nphase', nseq.getDefinition('Nphase'),...
                    'Nfreq', nseq.getDefinition('Nfreq'),...
                    'sfreq', nseq.getDefinition('gamma'),...
                    'tped', nseq.getDefinition('tped'), ...
                    'tramp', nseq.getDefinition('tramp'),...
                    'SW', nseq.getDefinition('spectralWidth'),...
                    'phi', nseq.getDefinition('phi'),...
                    'psi', nseq.getDefinition('psi'),...
                    'theta', nseq.getDefinition('theta'));
%% DELETE: Variables not present in current seq files:
```

```
% system_definitions.tped = 2E-3;
% system_definitions.tramp = 3E-3;
% system_definitions.sfreq = 42.577E6; % Hz/T
%% Defined System Constants:
  system_definitions.IDelay =  1.0E-7;  % Instrument Delay
  system_definitions.gamma = system_definitions.sfreq;
  system_definitions.ns = 1.0; % Number of slices, for future expansion into multislice this parameter can be
moved and defined in definitions
%% Identify the Primary Sequence:
% Declarations:
repeatedRF = 0;
blockcount = 0;
rfcount = 0;
gxcount = 0;
gycount = 0;
gzcount = 0;
adccount = 0;
numBE = length(nseq.blockEvents); % prevents error thrown
rfFlag = 0;

% Declare Sequence Loop Struct where the relevant info will be saved:
SequenceLoop = struct('delay', struct('block', [], 'duration',[], 'varName', []),...
   'rf', struct('block',[],'ID', [], 'duration', [], 'magnitudeID', [], 'phaseID', [],
'pattern',[],'width',[],'phase',[],'RG1',[], 'RG2', [], 'pulsepatname', [], 'obspowervar', [], 'phaseVarName', []),...
   'gx', struct('block', [], 'gradtype', [], 'duration', [], 'shapeID' ,[],'pattern',[], 'amplitude', [], 'width', [],
'riseTime', [], 'flatTime', [], 'fallTime', [], 'arbpatname', []),...
   'gy', struct('block', [], 'gradtype', [], 'duration', [],'shapeID' ,[],'pattern',[], 'amplitude', [], 'width', [],
'riseTime', [], 'flatTime', [], 'fallTime', [], 'arbpatname', []),...
   'gz', struct('block', [], 'gradtype', [], 'duration', [], 'shapeID' ,[],'pattern',[], 'amplitude', [], 'width', [],
'riseTime', [], 'flatTime', [], 'fallTime', [], 'arbpatname', []),...
   'adc', struct('block', [], 'ID', [], 'duration', [],'N', [], 'dwell', [], 'delay', []));

% Determine Length of the Pulse Sequence
while repeatedRF == 0
  blockcount = blockcount + 1;
  for i = 1:length(nseq.blockEvents{1,blockcount})
     % Check for RF pulses being repeated, if so, break to while
     if repeatedRF == 1
        break; % break to while loop, which should stop
     end
     b(blockcount) = nseq.getBlock(blockcount);
     % Switch based on the columns of the block
     switch i
      case 1 % Delay Case -- a block with delay will only have a delay
         if nseq.blockEvents{1,blockcount}(i) ~= 0
%              delaycount = blockcount + 1;
            SequenceLoop.delay(blockcount).block = blockcount;
            SequenceLoop.delay(blockcount).ID =nseq.blockEvents{1,blockcount}(1);
            SequenceLoop.delay(blockcount).duration= b(SequenceLoop.delay(blockcount).block).delay.delay;
            continue; % because no other ids will be present
         elseif nseq.blockEvents{1,blockcount}(i) == 0
            SequenceLoop.delay(blockcount).block = [];
            SequenceLoop.delay(blockcount).ID =[];
```

```
            SequenceLoop.delay(blockcount).duration= [];
          end
        case 2 % RF Case -- Look for Repeated IDs
          if nseq.blockEvents{1,blockcount}(i) ~= 0
          % IF condition to bypass first RF Pulse
            if  rfFlag ~=0
              % FOR the length of the vector containing the RF
              % Pulse IDs (variable length)
              rfterms = size(SequenceLoop.rf);
              for j = 1:rfterms(2) % number of unique rf pulses identified
                % IF this ID matches that of an existing ID
                if nseq.blockEvents{1,blockcount}(i) == SequenceLoop.rf(j).ID
                  % Check for repeated Pulses in a sequence
                  % prior to the first acquisition (inversion
                  % SE type cases, etc.)
                  adcterms = size(SequenceLoop.adc);
                  if ~isempty(SequenceLoop.adc(1:adcterms(2)))  % Check for at least one acquisition
                    repeatedRF = 1;
                    break; % breaks out of for loop for length of identified RF blocks to following if
repeatedRF == 1, break out of main for loop.
                  end % IF there has been at least one acquisition identified
              % ELSE: do nothing
                end % IF current rf ID matches an existing
              end  %FOR length of rf pulses identified so far
            end % IF
            % Check to break out of for block loop, otherwise
            % repeated RF saved
            if repeatedRF == 1
              break;
            end % IF Rf pulse repeated
            % if the RF ID is not present
            rfcount = rfcount + 1; % add one to the count
            SequenceLoop.rf(blockcount).obspowervar = 'var_tpwr' + string(rfcount);
            rfFlag = 1;
            SequenceLoop.rf(blockcount).block = blockcount; % add new term present at block
            SequenceLoop.rf(blockcount).ID = nseq.blockEvents{1,blockcount}(2);
            SequenceLoop.rf(blockcount).magnitudeID =
nseq.rfLibrary.data(SequenceLoop.rf(blockcount).ID).array(2);
            TEMPMAGID = SequenceLoop.rf(blockcount).magnitudeID;
            SequenceLoop.rf(blockcount).phaseID =
nseq.rfLibrary.data(SequenceLoop.rf(blockcount).ID).array(3);
            TEMPPHASEID = SequenceLoop.rf(blockcount).phaseID;
            SequenceLoop.rf(blockcount).width= max(b(blockcount).rf.t);
            SequenceLoop.rf(blockcount).duration = SequenceLoop.rf(blockcount).width;
            SequenceLoop.rf(blockcount).pulsepatname = '"sinc"';
            SequenceLoop.rf(blockcount).phaseVarName = 'v'+string(rfcount);

            tempStruct = struct('data',[],'num_samples',[]);
                tempStruct.data =
nseq.shapeLibrary.data(TEMPMAGID).array(2:length(nseq.shapeLibrary.data(TEMPMAGID).array));
                tempStruct.num_samples = nseq.shapeLibrary.data(TEMPMAGID).array(1);
            SequenceLoop.rf(blockcount).pattern = mr.decompressShape(tempStruct); % DECOMPRESSED
```

```
            % Need to add in a comparison for other existing shapes, if
            % shape is already existing in library, declare p1pat =
            % that shape name.  else, generate new shape name.

                tempStruct.data =
nseq.shapeLibrary.data(TEMPPHASEID).array(2:length(nseq.shapeLibrary.data(TEMPPHASEID).array));
                tempStruct.num_samples = nseq.shapeLibrary.data(TEMPPHASEID).array(1);
            SequenceLoop.rf(blockcount).phase= mr.decompressShape(tempStruct);

            SequenceLoop.rf(blockcount).RG1 = 'rof1'; % USER: FLAG: these are defined at system to be 10us
            SequenceLoop.rf(blockcount).RG2 = 'rof2'; % USER: FLAG: these are defined at system to be 10us


          elseif nseq.blockEvents{1,blockcount}(i) == 0
            SequenceLoop.rf(blockcount).block = [];
            SequenceLoop.rf(blockcount).ID = [];
            SequenceLoop.rf(blockcount).duration = [];
            SequenceLoop.rf(blockcount).magnitudeID = [];
            SequenceLoop.rf(blockcount).phaseID =  [];
            SequenceLoop.rf(blockcount).width = [];
            SequenceLoop.rf(blockcount).pattern = [];
            SequenceLoop.rf(blockcount).phase = [];
            SequenceLoop.rf(blockcount).pulsepatname = [];
            SequenceLoop.rf(blockcount).RG1 = []; % USER: FLAG: these are defined at system to be 10us
            SequenceLoop.rf(blockcount).RG2 = [];
          end % IF their is an RF block present
        case 3 % GX Case -- Identify GX parameters
          if nseq.blockEvents{1,blockcount}(i) ~= 0
            gxcount = gxcount+1;
            SequenceLoop.gx(blockcount).block = blockcount;
            SequenceLoop.gx(blockcount).ID = nseq.blockEvents{1,blockcount}(3);
%             SequenceLoop.gx(blockcount).type = b(SequenceLoop.gx(blockcount).block).gx.type;
            switch b(SequenceLoop.gx(blockcount).block).gx.type
              case 'trap'
                SequenceLoop.gx(blockcount).gradtype = 'trap';
                SequenceLoop.gx(blockcount).amplitude =
(b(SequenceLoop.gx(blockcount).block).gx.amplitude)/system_definitions.gamma * 100; %FLAG: need to
Convert
                SequenceLoop.gx(blockcount).riseTime = b(SequenceLoop.gx(blockcount).block).gx.riseTime;
                SequenceLoop.gx(blockcount).flatTime = b(SequenceLoop.gx(blockcount).block).gx.flatTime;
                SequenceLoop.gx(blockcount).fallTime = b(SequenceLoop.gx(blockcount).block).gx.fallTime;
                SequenceLoop.gx(blockcount).duration = SequenceLoop.gx(blockcount).riseTime +
SequenceLoop.gx(blockcount).flatTime + SequenceLoop.gx(blockcount).fallTime;

              case 'grad'
                SequenceLoop.gx(blockcount).gradtype = 'grad';
                SequenceLoop.gx(blockcount).pattern =
(b(SequenceLoop.gx(blockcount).block).gx.waveform)/max(b(SequenceLoop.gx(blockcount).block).gx.wavef
orm);
                SequenceLoop.gx(blockcount).amplitude =
max(b(SequenceLoop.gx(blockcount).block).gx.waveform)/system_definitions.gamma * 100;
                SequenceLoop.gx(blockcount).width =  b(SequenceLoop.gx(blockcount).block).gx.t;
                SequenceLoop.gx(blockcount).duration = SequenceLoop.gx(blockcount).width;
```

```
                SequenceLoop.gx(blockcount).arbpatname = 'shape'+string(gxcount)+'name';
            end
        elseif nseq.blockEvents{1,blockcount}(i) == 0
            SequenceLoop.gx(blockcount).block = [];
            SequenceLoop.gx(blockcount).ID = [];
            SequenceLoop.gx(blockcount).duration = [];
            SequenceLoop.gx(blockcount).type = [];
            SequenceLoop.gx(blockcount).amplitude = [];
            SequenceLoop.gx(blockcount).riseTime = [];
            SequenceLoop.gx(blockcount).flatTime = [];
            SequenceLoop.gx(blockcount).fallTime = [];
            SequenceLoop.gx(blockcount).pattern = [];
            SequenceLoop.gx(blockcount).width =  [];
            SequenceLoop.gx(blockcount).arbpatname = [];

        end
    case 4 % GY Case -- Identify GY parameters
        if nseq.blockEvents{1,blockcount}(i) ~= 0
            gycount = gycount+1;
            SequenceLoop.gy(blockcount).block = blockcount;
            SequenceLoop.gy(blockcount).ID = nseq.blockEvents{1,blockcount}(3);
            switch b(SequenceLoop.gy(blockcount).block).gy.type
                case 'trap'
                    SequenceLoop.gy(blockcount).gradtype = 'trap';
                    SequenceLoop.gy(blockcount).amplitude =
(b(SequenceLoop.gy(blockcount).block).gy.amplitude)/system_definitions.gamma * 100; %FLAG: need to
Convert
                    SequenceLoop.gy(blockcount).riseTime = b(SequenceLoop.gy(blockcount).block).gy.riseTime;
                    SequenceLoop.gy(blockcount).flatTime = b(SequenceLoop.gy(blockcount).block).gy.flatTime;
                    SequenceLoop.gy(blockcount).fallTime = b(SequenceLoop.gy(blockcount).block).gy.fallTime;
                    SequenceLoop.gy(blockcount).duration = SequenceLoop.gy(blockcount).riseTime +
SequenceLoop.gy(blockcount).flatTime + SequenceLoop.gy(blockcount).fallTime;

                case 'grad'
                    SequenceLoop.gy(blockcount).gradtype = 'grad';
                    SequenceLoop.gy(blockcount).pattern =
(b(SequenceLoop.gy(blockcount).block).gy.waveform)/max(b(SequenceLoop.gy(blockcount).block).gy.wave
form);
                    SequenceLoop.gy(blockcount).amplitude =
max(b(SequenceLoop.gy(blockcount).block).gy.waveform)/system_definitions.gamma * 100;
                    SequenceLoop.gy(blockcount).width =  b(SequenceLoop.gy(blockcount).block).gy.t;
                    SequenceLoop.gy(blockcount).duration = SequenceLoop.gy(blockcount).width;
                    SequenceLoop.gy(blockcount).arbpatname = 'shape'+string(gycount)+'name';
            end
        elseif nseq.blockEvents{1,blockcount}(i) == 0
            SequenceLoop.gy(blockcount).block = [];
            SequenceLoop.gy(blockcount).ID = [];
            SequenceLoop.gy(blockcount).duration = [];
            SequenceLoop.gy(blockcount).type = [];
            SequenceLoop.gy(blockcount).amplitude = [];
            SequenceLoop.gy(blockcount).riseTime = [];
            SequenceLoop.gy(blockcount).flatTime = [];
            SequenceLoop.gy(blockcount).fallTime = [];
```

```
            SequenceLoop.gy(blockcount).pattern = [];
            SequenceLoop.gy(blockcount).width =  [];
             SequenceLoop.gy(blockcount).arbpatname = [];
        end
    case 5 % GZ Case -- Identify GZ parameters
        if nseq.blockEvents{1,blockcount}(i) ~= 0
            gzcount = gzcount+1;
            SequenceLoop.gz(blockcount).block = blockcount;
            SequenceLoop.gz(blockcount).ID = nseq.blockEvents{1,blockcount}(3);
            switch b(SequenceLoop.gz(blockcount).block).gz.type;
                case 'trap'
                    SequenceLoop.gz(blockcount).gradtype = 'trap';
                    SequenceLoop.gz(blockcount).amplitude =
(((b(SequenceLoop.gz(blockcount).block).gz.amplitude))/(system_definitions.gamma))*100; %FLAG: need
to Convert
                    SequenceLoop.gz(blockcount).riseTime = b(SequenceLoop.gz(blockcount).block).gz.riseTime;
                    SequenceLoop.gz(blockcount).flatTime = b(SequenceLoop.gz(blockcount).block).gz.flatTime;
                    SequenceLoop.gz(blockcount).fallTime = b(SequenceLoop.gz(blockcount).block).gz.fallTime;
                    SequenceLoop.gz(blockcount).duration = SequenceLoop.gz(blockcount).riseTime +
SequenceLoop.gz(blockcount).flatTime + SequenceLoop.gz(blockcount).fallTime;

                case 'grad'
                    SequenceLoop.gz(blockcount).gradtype = 'grad';
                    SequenceLoop.gz(blockcount).pattern =
(b(SequenceLoop.gz(blockcount).block).gz.waveform)/max(b(SequenceLoop.gz(blockcount).block).gz.wavef
orm);
                    SequenceLoop.gz(blockcount).amplitude =
max(b(SequenceLoop.gz(blockcount).block).gz.waveform)/system_definitions.gamma * 100;
                    SequenceLoop.gz(blockcount).width =  b(SequenceLoop.gz(blockcount).block).gz.t;
                    SequenceLoop.gz(blockcount).duration = SequenceLoop.gz(blockcount).width;
                    SequenceLoop.gz(blockcount).arbpatname = 'shape'+string(gzcount)+'name';

            end
        elseif nseq.blockEvents{1,blockcount}(i) == 0
            SequenceLoop.gz(blockcount).block = [];
            SequenceLoop.gz(blockcount).ID = [];
            SequenceLoop.gz(blockcount).duration = [];
            SequenceLoop.gz(blockcount).type = [];
            SequenceLoop.gz(blockcount).amplitude = [];
            SequenceLoop.gz(blockcount).riseTime = [];
            SequenceLoop.gz(blockcount).flatTime = [];
            SequenceLoop.gz(blockcount).fallTime = [];
            SequenceLoop.gz(blockcount).pattern = [];
            SequenceLoop.gz(blockcount).width =  [];
            SequenceLoop.gz(blockcount).arbpatname = [];
        end
    case 6 % ADC Case -- Identify ADC parameters
        if nseq.blockEvents{1,blockcount}(i) ~= 0
            SequenceLoop.adc(blockcount).block = blockcount;
            SequenceLoop.adc(blockcount).ID = nseq.blockEvents{1,blockcount}(6);
            SequenceLoop.adc(blockcount).N = b(SequenceLoop.adc(blockcount).block).adc.numSamples;
            SequenceLoop.adc(blockcount).dwell = b(SequenceLoop.adc(blockcount).block).adc.dwell;
            SequenceLoop.adc(blockcount).delay = b(SequenceLoop.adc(blockcount).block).adc.delay;
```

```
            SequenceLoop.adc(blockcount).duration =
SequenceLoop.adc(blockcount).dwell*SequenceLoop.adc(blockcount).N;

          elseif nseq.blockEvents{1,blockcount}(i) == 0
            SequenceLoop.adc(blockcount).block = [];
            SequenceLoop.adc(blockcount).ID = [];
            SequenceLoop.adc(blockcount).N = [];
            SequenceLoop.adc(blockcount).dwell = [];
            SequenceLoop.adc(blockcount).delay = [];
            SequenceLoop.adc(blockcount).duration = [];
          end
      end
    end
end

b(blockcount).rf = [];
b(blockcount).gx = [];
b(blockcount).gy = [];
b(blockcount).gz = [];
b(blockcount).adc = [];
b(blockcount).delay = [];

SequenceLength = blockcount - 1;

for i = SequenceLength:-1:1
  if ~isempty(b(i).delay)
     condition(i) = "1 ";
   elseif isempty(b(i).delay)
     condition(i) = "0 ";
   end
   if ~isempty(b(i).rf)
     condition(i) = condition(i) + "1 ";
   elseif isempty(b(i).rf)
     condition(i) = condition(i) + "0 ";
   end
   if ~isempty(b(i).gx)
     condition(i) = condition(i) + "1 ";
   elseif isempty(b(i).gx)
     condition(i) = condition(i) + "0 ";
   end
   if ~isempty(b(i).gy)
     condition(i) = condition(i) + "1 ";
   elseif isempty(b(i).gy)
     condition(i) = condition(i) + "0 ";
   end
   if ~isempty(b(i).gz)
     condition(i) = condition(i) + "1 ";
   elseif isempty(b(i).gz)
     condition(i) = condition(i) + "0 ";
   end
   if ~isempty(b(i).adc)
     condition(i) = condition(i) + "1";
   elseif isempty(b(i).adc)
```

```
      condition(i) = condition(i) + "0";
   end
end
%% Determine Phase Encoding Parameters:
gycounter = 0;
for z = 1:numBE
   if nseq.blockEvents{1,z}(4) ~= 0
      gycounter = gycounter + 1;
      temp1 = nseq.getBlock(z);
      if temp1.gy.type == 'trap'
         temp2(gycounter) = (temp1.gy.amplitude/system_definitions.gamma) * 100 ;
      elseif temp1.gy.type == 'grad'
         temp2(gycounter) = (max(temp1.gy.waveform)/system_definitions.gamma) * 100;
      end
   end
end
% Find the differences between elements
% Flag: This process only works for cases with a single dynamic
% phase encoding pulse.  Need to expand to detect multiple dynamic pulses.

for z = 2:(length(temp2))
   diff(z-1) = temp2(z) - temp2(z-1);
end
gpeBaseVal = min(temp2);
gpeStepVal = mode(diff);
%% Begin Output File:
completedFile = 0;
while completedFile == 0
%% Preamble
fileID = fopen(filename,'w');
preamblea = '/* This Varian-Compatible Pulse Sequence File was created ';
preambleb = '   using TAMU MRSLs Matlab to Varian-C translator.       */';
nl = '\n';
fprintf(fileID, '%1$s\n%2$s\n', preamblea,preambleb);
fprintf(fileID, nl);
clear preamblea preambleb;
%% Declarations
fprintf(fileID, '%1$s\n', '#include <standard.h>');
fprintf(fileID, nl);
% May put Phase Encoding Gradient Array for tables here

% Minimum instrument delay
fprintf(fileID, '%1$s %2$1.5e\n', '#define ID', system_definitions.IDelay);
fprintf(fileID, '%1$s %2$8.6f\n', '#define gpeBase', gpeBaseVal);
fprintf(fileID, '%1$s %2$8.6f\n', '#define gpeStep', gpeStepVal);

fprintf(fileID, nl);

% Phase Tables
fprintf(fileID,
'%1$s%2$s%3$c%4$s%5$s\n\t\t\t%2$s%6$c%4$s%7$s\n\t\t\t%2$s%8$c%4$s%9$s\n\t\t\t%2$s%10$c%4$
s%11$s\n\n',...
            'static int ','ph','1', '[4] = {','0,2,1,3},',...
```

```
                '2','1,1,2,2},',...
                '3','3,3,0,0},',...
                'r','0,2,1,3};');

% Counter Variable for Gpe Calculation
fprintf(fileID, '%1$s %2$2.1f;\n\n', 'static double counter =', 0.0);
%% Start Pulse Sequence
fprintf(fileID, '%1$s\n%2$s\n', 'pulsesequence()', '{');

% Write Variable Declarations
fprintf(fileID, '%1$s  %2$s, %3$s, %4$s, %5$s;\n', ...
                  'double', 'pi2', 'gamma', 'spectralWidth');
fprintf(fileID, nl);
%fprintf(fileID, '%1$s  %2$s, %3$s;\n', ...
 %                   'double', 'tpwr1', 'tpwr2');

 % FLAG: Create for loop that declares RF pulse power variables

if rfcount >= 2
   rfcounter = 0;
   fprintf(fileID, '%1$s ', 'double');
  for a = 1:1:SequenceLength
     if ~isempty(SequenceLoop.rf(a).block)
       rfcounter = rfcounter + 1;
        if rfcounter < rfcount
        fprintf(fileID, '%1$s, ',  SequenceLoop.rf(a).obspowervar);
        elseif rfcounter == rfcount
          fprintf(fileID, '%1$s;\n',  SequenceLoop.rf(a).obspowervar);
        end
     end
  end
elseif rfcount == 1
   fprintf(fileID, '%1$s %2$s;\n', 'double', SequenceLoop.rf(1).obspowervar);
end

fprintf(fileID, nl);

fprintf(fileID,'%1$s;\n', 'initparms_sis()'); % may need to remove this in order to bypass values
fprintf(fileID, nl);

fprintf(fileID, nl);


fprintf(fileID, '%1$s = %2$4.2f/%3$2.1f;\n', 'pi2', 3.14, 2.0);
fprintf(fileID, '%1$s = %2$8.2e; /*%3$s*/\n', 'gamma', system_definitions.gamma, ' Replacing: gamma =
sfrq*1e6/B0;');

if rfcount >= 2
   rfcounter = 0;
  for a = 1:1:SequenceLength
     if ~isempty(SequenceLoop.rf(a).block)
       rfcounter = rfcounter + 1;
        if rfcounter < rfcount
```

```
        fprintf(fileID, '%1$s = %2$s;\n', SequenceLoop.rf(a).obspowervar, '/*FLAG: INSERT CALIBRATED FIRST
PULSE POWER HERE*/');
        elseif rfcounter == rfcount
        fprintf(fileID, '%1$s = %2$s;\n', SequenceLoop.rf(a).obspowervar, '/*FLAG: INSERT CALIBRATED FIRST
PULSE POWER HERE*/');
        end
      end
  end
elseif rfcount == 1
   fprintf(fileID, '%1$s = %2$s;\n', SequenceLoop.rf(a).obspowervar, '/*FLAG: INSERT CALIBRATED FIRST
PULSE POWER HERE*/');
end

fprintf(fileID, '%1$s = %2$5.1f;\n', 'nv', system_definitions.Nphase);
fprintf(fileID, '%1$s = %2$5.4e;\n', 'spectralWidth', system_definitions.SW);
fprintf(fileID, nl);

fprintf(fileID, '%1$s = %2$5.2f;\n', 'theta', system_definitions.theta);
fprintf(fileID, '%1$s = %2$5.2f;\n', 'phi', system_definitions.phi);
fprintf(fileID, '%1$s = %2$5.2f;\n', 'psi', system_definitions.psi);
fprintf(fileID, nl);

% Write Phase Looping Declarations
% nv is the number of phase encode steps, so replacing the following line with the line below it
% fprintf(fileID, '%1$s(%2$s/%3$2.1f,%4$s);\n', 'initval', 'nv', 2.0, 'v7');
fprintf(fileID, '%1$s(%2$2.1f/%3$2.1f,%4$s);\n', 'initval', system_definitions.Nphase, 2.0, 'v7');
fprintf(fileID, '%1$s(%2$s,%3$d,%4$s);\n', 'settable', 't1', 4, 'ph1');
fprintf(fileID, nl);

fprintf(fileID, '%1$s(%2$s,%3$s,%4$s);\n', 'getelem', 't1', 'ct', 'v1');
fprintf(fileID, '%1$s(%2$s,%3$d,%4$s);\n', 'settable', 't2', 4, 'ph2');
fprintf(fileID, '%1$s(%2$s,%3$d,%4$s);\n', 'settable', 't3', 4, 'ph3');
fprintf(fileID, '%1$s(%2$s,%3$d,%4$s);\n', 'settable', 't4', 4, 'phr');
fprintf(fileID, '%1$s(%2$s,%3$s,%4$s);\n', 'getelem', 't4', 'ct', 'oph');
fprintf(fileID, nl);


fprintf(fileID,'%1$s\n','/* PULSE SEQUENCE ************************************/');
fprintf(fileID, '%1$s(%2$s);\n', 'status', 'A');
fprintf(fileID, '%1$s(%2$s);\n', 'obspower', 'var_tpwr1');  % Hard coded because normally re-declared prior
to pulse
fprintf(fileID, nl);

% FLAG: Add Safety Checks to max gradient values

fprintf(fileID, '%1$s(%2$s[%3$d],%4$s,%5$s,%6$s);\n', 'peloop', 'seqcon', 2, 'nv', 'v5', 'v6');
%fprintf(fileID, '\t%1$s(%2$s[%3$d],%4$2.1f,%5$s,%6$s);\n', 'msloop', 'seqcon', 1, system_definitions.ns ,
'v11', 'v12');  % FLAG: for expansion into multslice, make this dynamic, for now, ns = 1;
fprintf(fileID, nl);

fprintf(fileID,'%1$s\n','/* Phase Cycling ************************************/');
fprintf(fileID, '\t%1$s(%2$s,%3$s);\n', 'mod2', 'v6', 'v4');
fprintf(fileID, '\t%1$s(%2$s);\n', 'ifzero', 'v4');
```

```
fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s);\n', 'getelem', 't2', 'ct', 'v2');
fprintf(fileID, '\t%1$s(%2$s);\n', 'elsenz', 'v4');
fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s);\n', 'getelem', 't3', 'ct', 'v2');
fprintf(fileID, '\t%1$s(%2$s);\n', 'endif', 'v4');
fprintf(fileID, nl);

fprintf(fileID,'%1$s\n','/* Phase Encode Gradient Amp Calculation **********/');

fprintf(fileID, nl);

fprintf(fileID, '\t%1$s(%2$s,%3$s,%4$2.1f,%5$s);\n', 'poffset_list', 'pss', 'gss', system_definitions.ns,'v12'); %
PSS is slice position, gss is gradient strength at that position, ns is number of slices in multislice,



for i = 1:SequenceLength
   fprintf(fileID,'/* Block Number %1$d */\n ', i);

   temp = condition(i);

   structFields = fieldnames(SequenceLoop)';
   % Organize and resturcture
   active = strsplit(temp);
   a = 0;
   for j = length(structFields):-1:1
      if active(j) ~= "0"
         a = a+1;
         relevantFields(a) = string(structFields{1,j});
      end
   end
   gradsInBlock = 0;
   % Record durations of each event in block
   for j = 1:length(relevantFields)
      durations(j) = struct('event', [relevantFields(j)], 'd', [SequenceLoop.(char(relevantFields(j)))(i).duration],
'type', []);
      if (durations(j).event == 'gx') || (durations(j).event == 'gy') || (durations(j).event == 'gz')
         grad = string(durations(j).event);
         gradsInBlock = gradsInBlock+1;
         switch grad
            case 'gx'
               durations(j).type = SequenceLoop.gx(i).gradtype;
            case 'gy'
               durations(j).type = SequenceLoop.gy(i).gradtype;
            case 'gz'
               durations(j).type = SequenceLoop.gz(i).gradtype;
         end
      else
         durations(j).type = '';
      end

   end

   % Make Struct to Cell Array and Sort Events from Longest
```

```
% Duration to Shortest
dcell = struct2cell(durations);
[m1,m2,m3] = size(dcell);
dcell = reshape(dcell, m1, []);
dcell = dcell';
dcell = cell2table(dcell);
dcell = sortrows(dcell,'dcell1','ascend');

rfPresent = 0;
adcPresent = 0;

tol = eps(0.7);

for z = 1:size(dcell,1)
   if dcell{z,1} == 'rf'
      fprintf(fileID, '\t\t%1$s(%2$s);\n', 'obspower', string(SequenceLoop.rf(i).obspowervar));
      rfPresent = 1;
   end
   if dcell{z,1} == 'adc'
      fprintf(fileID, '\t\t%1$s(%2$s,%3$s);\n', 'poffset', 'pro', 'gro');
      adcPresent = 1;
   end
end

 % Begin Writing Block
for j = 1:size(dcell,1)
    switch char([dcell{j,1}])
       case 'delay'
          fprintf(fileID, '\t\t%1$s(%2$8.7f+%3$s);\n', 'delay',SequenceLoop.delay(i).duration , 'ID');
       case 'rf'
          if gradsInBlock > 0
             % do nothing
          else
             fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shapedpulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration, 'v1',string(SequenceLoop.rf(i).RG1),
string(SequenceLoop.rf(i).RG2));
          end
       case {'gx', 'gy', 'gz'}
          switch gradsInBlock
             case 1
                if dcell{j,1} == 'gx'
                  oneGradient(SequenceLoop,dcell,i,j,fileID, 'gx', rfPresent, adcPresent,tol)
                end
                if dcell{j,1} == 'gy'
                  fprintf(fileID,'\t\tgpeAmp = gpeBase + (counter * gpeStep);\n ');
                     oneGradient(SequenceLoop,dcell,i,j,fileID, 'gy', rfPresent, adcPresent,tol)
                end
                if dcell{j,1} == 'gz'
                     oneGradient(SequenceLoop,dcell,i,j,fileID, 'gz', rfPresent, adcPresent,tol)
                end
             case 2
                if dcell{j,1} == 'gx'
                   for z = 1:size(dcell,1)
```

```
                    if dcell{z,1} == 'gy'
                        dcell{z,1} = ' ';
                        fprintf(fileID,'\t\tgpeAmp = gpeBase + (counter * gpeStep);\n ');
                        twoGradients(SequenceLoop,i,fileID, 'gx', 'gy', rfPresent, adcPresent,tol);
                    elseif dcell{z,1} == 'gz'
                        dcell{z,1} = ' ';
                        twoGradients(SequenceLoop,i,fileID, 'gx', 'gz', rfPresent, adcPresent,tol);

                    end
                  end
                end
                if dcell{j,1} == 'gy'
                  fprintf(fileID,'\t\tgpeAmp = gpeBase + (counter * gpeStep);\n ');
                  for z = 1:size(dcell,1)
                    if dcell{z,1} == 'gx'
                      dcell{z,1} = ' ';
                      twoGradients(SequenceLoop,i,fileID, 'gy', 'gx', rfPresent, adcPresent,tol);
                    elseif dcell{z,1} == 'gz'
                      dcell{z,1} = ' ';
                      twoGradients(SequenceLoop,i,fileID, 'gy', 'gz', rfPresent, adcPresent,tol);
                    end
                  end
                end
                if dcell{j,1} == 'gz'
                  for z = 1:size(dcell,1)
                    if dcell{z,1} == 'gx'
                      dcell{z,1} = ' ';
                      twoGradients(SequenceLoop,i,fileID, 'gz', 'gx', rfPresent, adcPresent,tol);
                    elseif dcell{z,1} == 'gy'
                      dcell{z,1} = ' ';
                      fprintf(fileID,'\t\tgpeAmp = gpeBase + (counter * gpeStep);\n ');
                      twoGradients(SequenceLoop,i,fileID, 'gz', 'gy', rfPresent, adcPresent,tol);
                    end
                  end
                end
                break;
              case 3
                fprintf(fileID,'\t\tgpeAmp = gpeBase + (counter * gpeStep);\n ');
                threeGradients(SequenceLoop, i, fileID, 'gx' , 'gy', 'gz', rfPresent, adcPresent, tol);
                break;
            end
          case 'adc'
            if gradsInBlock > 0
              % do nothing
            else
              fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
              fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
              fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            end
        end
    end
    clear dFields dcell sz temp1 durations relevantFields
```

```
end
%% Wrap Up End of File
fprintf(fileID, nl);
%fprintf(fileID, '\t%1$s(%2$s[%3$d],%4$s);\n', 'endmsloop', 'seqcon', 1, 'v12');  % FLAG: for expansion into
multslice, make this dynamic, for now, ns = 1;
fprintf(fileID, '%1$s(%2$s[%3$d],%4$s);\n', 'endpeloop', 'seqcon', 2, 'v6');
fprintf(fileID, nl);
fprintf(fileID, '}\n');
completedFile = 1;
end
%% Other Functions
% Read the .seq file using the Pulseq Reader
function seqData = readSeqFile(seqFile)
%readSeqFile
clear seqData;
seqData = mr.Sequence();  % Initialize
seqData.read(seqFile);    % Read in the Filename
end
%
%% End of Function
end
```

oneGradient.m

```
function [  ] = oneGradient(SequenceLoop,dcell,i,j,fileID, channel1, rfPresent, adcPresent,tol)


gx = struct('Shape1', [''],'Shape2', [''], 'Amp', 0);
gy = struct('Shape1', [''],'Shape2', [''], 'Amp', 0);
gz = struct('Shape1', [''],'Shape2', [''], 'Amp', 0);

turnOffGx = 0;
temp = SequenceLoop.(char(channel1))(i).gradtype;
switch temp
  case 'trap'
    if channel1 == "gx"
    gx.Shape1 = 'ramp_hold';
    gx.Shape2 = 'ramp_down';
    gx.Amp = SequenceLoop.gx(i).amplitude;
    else
    end

    if channel1 == "gy"
    gy.Shape1 = 'ramp_hold';
    gy.Shape2 = 'ramp_down';
    gy.Amp = 'gpeAmp';
    else
    gy.Amp = '    0';
    end
    if channel1 == "gz"
    gz.Shape1 = 'ramp_hold';
    gz.Shape2 = 'ramp_down';
    gz.Amp = SequenceLoop.gz(i).amplitude;
    else
    end

    if rfPresent == 1
       for z = 1:size(dcell,1)
          if dcell{z,1} == 'rf' %
             if (SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1))(i).flatTime > 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1))(i).flatTime) > tol % IF RF duration >
gradient duration
                warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
                fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
                fprintf(fileID, '\t\t/*%1$s(%2$s,%3$f,%4$5.4g,%5$s,%6$5.4g,%7$d,%8$s);*/ \n',
'obl_shapedgradient','custom shape file names: gro, gpe, gss', SequenceLoop.(char(channel1))(i).duration,
gx.Amp, gy.Amp, gz.Amp, 1, 'NOWAIT'); % FLAG CONVERT AMPLITUDES TO GCM
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1))(i).riseTime);
                fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shapedpulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
% FLAG MAY NEED TO CHANGE TO BREAK UP SHAPE SO THAT
```

```
            elseif abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1))(i).flatTime) < tol % IF RF
duration == gradient duration
                turnOffGx = 1;
                fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%6$5.4g,%7$s,%8$5.4g,%9$d,%10$s);\n',
'obl_shapedgradient', gx.Shape1, gy.Shape1, gz.Shape1, SequenceLoop.(char(channel1))(i).riseTime,
gx.Amp, gy.Amp, gz.Amp, 1, 'WAIT'); % FLAG CONVERT AMPLITUDES TO GCM
                fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shapedpulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
% FLAG MAY NEED TO CHANGE TO BREAK UP SHAPE SO THAT
                % QUESTION: CHECK THESE
                % CASES

            elseif (SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1))(i).flatTime < 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1))(i).flatTime) > tol % IF RF duration <
gradient duration
                turnOffGx = 1;
                fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%6$8.f,%7$s,%8$f,%9$d,%10$s);\n',
'obl_shapedgradient', gx.Shape1, gy.Shape1, gz.Shape1, SequenceLoop.(char(channel1))(i).riseTime,
gx.Amp, gy.Amp, gz.Amp, 1, 'WAIT'); % FLAG CONVERT AMPLITUDES TO GCM
                fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shapedpulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(SequenceLoop.(char(channel1))(i).flatTime -
SequenceLoop.rf(i).duration));
            end
          end
       end

   elseif adcPresent == 1
       if rfPresent == 1
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
          fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
       else
          if ((SequenceLoop.adc(i).duration - SequenceLoop.(char(channel1))(i).flatTime) > 0) &&
(abs(SequenceLoop.adc(i).duration - SequenceLoop.(char(channel1))(i).flatTime) > tol)
            warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
            fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
            fprintf(fileID, '\t\t/*%1$s(%2$s,%3$f,%4$5.4g,%5$s,%6$5.4g,%7$d,%8$s);*/ \n',
'obl_shapedgradient','custom shape file names: gro, gpe, gss', SequenceLoop.(char(channel1))(i).duration,
gx.Amp, gy.Amp, gz.Amp, 1, 'NOWAIT'); % FLAG CONVERT AMPLITUDES TO GCM
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1))(i).riseTime);

            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

         elseif abs(SequenceLoop.adc(i).duration - SequenceLoop.(char(channel1))(i).flatTime) < tol % IF RF
duration == gradient duration
```

```
            turnOffGx = 1;
            fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%6$5.4g,%7$s,%8$5.4g,%9$d,%10$s);\n',
'obl_shapedgradient', gx.Shape1, gy.Shape1, gz.Shape1, SequenceLoop.(char(channel1))(i).riseTime,
gx.Amp, gy.Amp, gz.Amp, 1, 'WAIT'); % FLAG CONVERT AMPLITUDES TO GCM
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

         elseif (SequenceLoop.adc(i).duration - (SequenceLoop.(char(channel1))(i).flatTime)  < 0) &&
abs(SequenceLoop.adc(i).duration - SequenceLoop.(char(channel1))(i).flatTime) > tol % IF RF duration <
gradient duration
            turnOffGx = 1;
            fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%6$5.4g,%7$s,%8$5.4g,%9$d,%10$s);\n',
'obl_shapedgradient', gx.Shape1, gy.Shape1, gz.Shape1, SequenceLoop.(char(channel1))(i).riseTime,
gx.Amp, gy.Amp, gz.Amp, 1, 'WAIT'); % FLAG CONVERT AMPLITUDES TO GCM
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(SequenceLoop.(char(channel1))(i).flatTime) -
SequenceLoop.adc(i).duration);
         end
      end
   else
      turnOffGx = 1;
      fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%6$5.4g,%7$s,%8$5.4g,%9$d,%10$s);\n',
'obl_shapedgradient', gx.Shape1, gy.Shape1, gz.Shape1, SequenceLoop.(char(channel1))(i).riseTime,
gx.Amp, gy.Amp, gz.Amp, 1, 'WAIT'); % FLAG CONVERT AMPLITUDES TO GCM
      fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(SequenceLoop.(char(channel1))(i).flatTime));
      %fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%7$f,%6$d,%6$d,%8$d,%9$s);\n',
'obl_shapedgradient', 'ramp_down', '', '', SequenceLoop.(char(channel1))(i).riseTime, 0,
SequenceLoop.(char(channel1))(i).amplitude, 1, 'WAIT'); % FLAG CONVERT AMPLITUDES TO GCM
   end
 case 'grad'

   if channel1 == "gx"
   gx.Shape1 = SequenceLoop.gx(i).arbpatname;
   gx.Amp = SequenceLoop.gx(i).amplitude;
   elseif channel1 == "gy"
   gy.Shape1 = SequenceLoop.gy(i).arbpatname;
   gy.Amp = SequenceLoop.gy(i).amplitude;
   elseif channel1 == "gz"
   gz.Shape1 = SequenceLoop.gz(i).arbpatname;
   gz.Amp = SequenceLoop.gz(i).amplitude;
   end

   if rfPresent == 1
      fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%6$5.4g,%7$s,%8$5.4g,%9$d,%10$s);\n',
'obl_shapedgradient', gx.Shape1, gy.Shape1, gz.Shape1, SequenceLoop.(char(channel1))(i).duration,
gx.Amp, gy.Amp, gz.Amp, 1, 'NOWAIT'); % FLAG CONVERT AMPLITUDES TO GCM
      fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shapedpulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
```

```
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
% FLAG MAY NEED TO CHANGE TO BREAK UP SHAPE SO THAT
   end
   if adcPresent == 1
      if rfPresent == 1
         % Need to look at this case and
         % how to simulatneously
         % transmit and receive.
         fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
         fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
         fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
      else
         fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%6$5.4g,%7$s,%8$5.4g,%9$d,%10$s);\n',
'obl_shapedgradient', gx.Shape1, gy.Shape1, gz.Shape1, SequenceLoop.(char(channel1))(i).duration,
gx.Amp, gy.Amp, gz.Amp, 1, 'NOWAIT'); % FLAG CONVERT AMPLITUDES TO GCM
         fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
         fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
         fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
      end
   else
      turnOffGx = 1;
      fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%6$5.4g,%7$s,%8$5.4g,%9$d,%10$s);\n',
'obl_shapedgradient', gx.Shape1, gy.Shape1, gz.Shape1, SequenceLoop.(char(channel1))(i).duration,
gx.Amp, gy.Amp, gz.Amp, 1, 'NOWAIT'); % FLAG CONVERT AMPLITUDES TO GCM
      fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(SequenceLoop.(char(channel1))(i).flatTime));
   end
end
if turnOffGx > 0
      fprintf(fileID, '\t\t%1$s("%2$s","%3$s","%4$s",%5$f,%6$5.4g,%7$s,%8$5.4g,%9$d,%10$s);\n',
'obl_shapedgradient', gx.Shape2, gy.Shape2, gz.Shape2, SequenceLoop.(char(channel1))(i).riseTime,
gx.Amp, gy.Amp, gz.Amp, 1, 'WAIT'); % FLAG CONVERT AMPLITUDES TO GCM
end
fprintf(fileID, '\t\t%1$s(%2$s);\n', 'zero_all_gradients', '');
end
```

twoGradients.m

```
function [  ] = twoGradients_sameDurGradMod(SequenceLoop,i,fileID, input1, input2, rfPresent,
adcPresent,tol)
%% Declarations
gradInputStart1 = '';
gradInputStart2 = '';
gradInputStartAmp1 = '';
gradInputStartAmp2 = '';
gradInputStop1 = '';
gradInputStop2 = '';
gradInputStopAmp1 = '';
gradInputStopAmp2 = '';

 %% Determine the channel with the longer gradient
   % This is made in an effort to streamline code cases.
   % Need to verify this is working properly, may need to add the
   % tolerance for floating point numbers
    if SequenceLoop.(char(input1))(i).gradtype == "trap"
       compVal1 = SequenceLoop.(char(input1))(i).flatTime;
    elseif SequenceLoop.(char(input1))(i).gradtype == "grad"
       compVal1 = SequenceLoop.(char(input1))(i).duration;
    end

    if SequenceLoop.(char(input2))(i).gradtype == "trap"
       compVal2 = SequenceLoop.(char(input2))(i).flatTime;
    elseif SequenceLoop.(char(input2))(i).gradtype == "grad"
       compVal2 = SequenceLoop.(char(input2))(i).duration;
    end

    if compVal1 >= compVal2
      channel1grad = input1;
      channel2grad = input2;
    elseif compVal1 < compVal2
      channel1grad = input2;
      channel2grad = input1;
    end

    if (SequenceLoop.(char(channel1grad))(i).gradtype == "trap")
       refDur = SequenceLoop.(char(channel1grad))(i).flatTime;
    elseif (SequenceLoop.(char(channel1grad))(i).gradtype == "grad")
       refDur = SequenceLoop.(char(channel1grad))(i).duration;
    end

%% Trapezoidal Case
if (SequenceLoop.(char(channel1grad))(i).gradtype == "trap") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "trap")
      % Same Duration Case
      % Building Start Gradient Input
         if channel1grad == "gx"
            gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
```

```
        elseif channel2grad == "gx"
           gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
           gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ',',
',});
        else
           gradInputStart1 = strjoin({gradInputStart1, '"",'});
           gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
        end
        if channel1grad == "gy"
           gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
           gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp, '});
        elseif channel2grad == "gy"
           gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
           gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp, '});
        else
           gradInputStart1 = strjoin({gradInputStart1, '"",'});
           gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
        end
        if channel1grad == "gz"
           gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
           gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
        elseif channel2grad == "gz"
           gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
           gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
        else
           gradInputStart1 = strjoin({gradInputStart1, '"",'});
           gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
        end

        % Building Stop Gradient Input Separately from Start because they may need to be separated later
on.
        if channel1grad == "gx"
           gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
           gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});

        elseif channel2grad == "gx"
           gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
           gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});

        else
           gradInputStop1 = strjoin({gradInputStop1, '"",'});
           gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
        end
        if channel1grad == "gy"
           gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
           gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp, '});
        elseif channel2grad == "gy"
           gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
           gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp, '});
        else
           gradInputStop1 = strjoin({gradInputStop1, '"",'});
           gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
        end
```

```
        if channel1grad == "gz"
           gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
           gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
        elseif channel2grad == "gz"
           gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
           gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
        else
           gradInputStop1 = strjoin({gradInputStop1, '""','});
           gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
        end

        if rfPresent == 1
           % IF RF duration > Gradient Duration
           if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
              fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
              fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
              fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).riseTime);
              fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

           % IF RF Duration == Gradient flat time
           elseif abs(SequenceLoop.rf(i).duration - refDur) < tol % IF RF duration == trap gradient flat time
              fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
              fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

           % IF RF Duration < Gradient Duration
           elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
% IF RF duration < gradient duration
              % IF RF pulse ends During Flat Time
                 fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
                 fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

           end
        end
        if adcPresent == 1
           % Gradient RF ADC Case
           if rfPresent == 1
              fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
              fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
              fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
```

```
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            % Do nothing Case

            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
               fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
               fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
               fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
         % Gradient ADC Case
         else
            % IF ADC Duration > Gradient Duration
            if ((SequenceLoop.adc(i).duration - refDur) > 0) && (abs(SequenceLoop.adc(i).duration - refDur)
> tol)
               warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
               fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after
a specific duration.*/\n');
               fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom
shape file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
               fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1))(i).riseTime);
               fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
               fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
               fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

            % IF ADC Duration == Gradient Duration
            elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
               fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
               fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
               fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
               fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
               fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            % IF ADC Duration < Gradient Duration
            elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.adc(i).duration -
refDur) > tol % IF RF duration < gradient duration
               fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
               fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
               fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
               fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
               fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(SequenceLoop.(char(channel1))(i).flatTime -
SequenceLoop.adc(i).duration));
```

```
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            end
          end

        elseif (adcPresent == 0) && (rfPresent == 0) % No RF or ADC
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur);
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');



        % Write Ramps Down
        elseif (adcPresent == 0) && (rfPresent == 1)
          % FLAG: Add Conditional to RF Turn off
          if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            % Do nothing Case
          elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

          elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          end
        end

end
%% Trapezoidal and Arbitrary Case
if (SequenceLoop.(char(channel1grad))(i).gradtype == "trap") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "grad")
  % Two Separate Channels for Starts due to differences in shapes
  % Building Start Gradient Input because all events in a block are assumed to start
    % at the same time
    if channel1grad == "gx"
      gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
      gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
    else
      gradInputStart1 = strjoin({gradInputStart1, '"",'});
      gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
    end
    if channel2grad == "gx"
      gradInputStart2 = strjoin({gradInputStart2,'"', string(SequenceLoop.gx(i).arbpatname),'",'});
      gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
    else
      gradInputStart2 = strjoin({gradInputStart2, '"",'});
      gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
    end
```

```
if channel1grad == "gy"
   gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp, '});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel2grad == "gy"
   gradInputStart2 = strjoin({gradInputStart2, '"', string(SequenceLoop.gy(i).arbpatname),'",'});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp, '});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel2grad == "gz"
   gradInputStart2 = strjoin({gradInputStart2,'"', string(SequenceLoop.gz(i).arbpatname),'",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude),','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

% Building Stop Gradient Input Separately
% Stops not needed for arbritary shape
if channel1grad == "gx"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
end
if channel1grad == "gy"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp, '});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
end
if channel1grad == "gz"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
end
```

```
    if rfPresent == 1
        % IF RF duration > Gradient Duration
        if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
          fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
          fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).riseTime);
          fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
% FLAG MAY NEED TO CHANGE TO BREAK UP SHAPE SO THAT

        % IF RF Duration == Gradient flat time
        elseif abs(SequenceLoop.rf(i).duration - refDur) < tol % IF RF duration == trap gradient flat time
          fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

        % IF RF Duration < Gradient Duration
        elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
% IF RF duration < gradient duration
          % IF RF pulse ends During Flat Time
            fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
        end
    end
    if adcPresent == 1
        % Gradient RF ADC Case
        if rfPresent == 1
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
          fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

          if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          % Do nothing Case

          elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
```

```
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
          % Gradient ADC Case
          else
            % IF ADC Duration > Gradient Duration
            if ((SequenceLoop.adc(i).duration - refDur) > 0) && (abs(SequenceLoop.adc(i).duration - refDur)
> tol)
                warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
                fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after
a specific duration.*/\n');
                fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom
shape file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1))(i).riseTime);
                fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

            % IF ADC Duration == Gradient Duration
            elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
                fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            % IF ADC Duration < Gradient Duration
            elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.adc(i).duration -
refDur) > tol % IF RF duration < gradient duration
                fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
```

```
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(SequenceLoop.(char(channel1))(i).flatTime -
SequenceLoop.adc(i).duration));
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            end
        end

    elseif (adcPresent == 0) && (rfPresent == 0) % No RF or ADC
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
        fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1, 'WAIT');
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');



    % Write Ramps Down
    elseif (adcPresent == 0) && (rfPresent == 1)
        % FLAG: Add Conditional to RF Turn off
        if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            % Do nothing Case
        elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

        elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
        end
    end
end
%% Arbitrary and Trapezoidal Case
  if (SequenceLoop.(char(channel1grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "trap")
  % Two Separate Channels for Starts due to differences in shapes
  % Building Start Gradient Input because all events in a block are assumed to start
    % at the same time
    if channel1grad == "gx"
      gradInputStart1 = strjoin({gradInputStart1,'"', string(SequenceLoop.gx(i).arbpatname),'",'});
      gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
    else
      gradInputStart1 = strjoin({gradInputStart1, '"",'});
      gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
    end
    if channel2grad == "gx"
      gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
      gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
    else
      gradInputStart2 = strjoin({gradInputStart2, '"",'});
      gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
```

```
end

if channel1grad == "gy"
   gradInputStart1 = strjoin({gradInputStart1,'"', string(SequenceLoop.gy(i).arbpatname),'",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp, '});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel2grad == "gy"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp, '});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1,'"', string(SequenceLoop.gz(i).arbpatname),'",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel2grad == "gz"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude),','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

% Building Stop Gradient Input Separately
% Stops not needed for arbritary shape
if channel2grad == "gx"
   gradInputStop2 = strjoin({gradInputStop2, '"ramp_down",'});
   gradInputStopAmp2 = strjoin({gradInputStopAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStop2 = strjoin({gradInputStop2, '"",'});
   gradInputStopAmp2 = strjoin({gradInputStopAmp2, '0,'});
end
if channel2grad == "gy"
   gradInputStop2 = strjoin({gradInputStop2, '"ramp_down",'});
   gradInputStopAmp2 = strjoin({gradInputStopAmp2, 'gpeAmp, '});
else
   gradInputStop2 = strjoin({gradInputStop2, '"",'});
   gradInputStopAmp2 = strjoin({gradInputStopAmp2, '0,'});
end
if channel2grad == "gz"
   gradInputStop2 = strjoin({gradInputStop2, '"ramp_down",'});
   gradInputStopAmp2 = strjoin({gradInputStopAmp2, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStop2 = strjoin({gradInputStop2, '"",'});
```

```
        gradInputStopAmp2 = strjoin({gradInputStopAmp2, '0,'});
    end

    if rfPresent == 1
        % IF RF duration > Gradient Duration
        if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
          fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient','custom shape file
names: gro, gpe, gss', SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay', SequenceLoop.(char(channel2))(i).riseTime);
          fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n',
'obl_shapedgradient',gradInputStart1, refDur, gradInputStartAmp1, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
% FLAG MAY NEED TO CHANGE TO BREAK UP SHAPE SO THAT

        % IF RF Duration == Gradient flat time
        elseif abs(SequenceLoop.rf(i).duration - refDur) < tol % IF RF duration == trap gradient flat time
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

        % IF RF Duration < Gradient Duration
        elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
% IF RF duration < gradient duration
          % IF RF pulse ends During Flat Time
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
        end
    end
    if adcPresent == 1
        % Gradient RF ADC Case
        if rfPresent == 1
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
          fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

          if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          % Do nothing Case
```

```
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStopAmp2, 1, 'WAIT');

            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStopAmp2, 1, 'WAIT');
            end


        % Gradient ADC Case
        else
            % IF ADC Duration > Gradient Duration
            if ((SequenceLoop.adc(i).duration - refDur) > 0) && (abs(SequenceLoop.adc(i).duration - refDur)
> tol)
                warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
                fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after
a specific duration.*/\n');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1,
'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel2))(i).riseTime);
                fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n',
'obl_shapedgradient',gradInputStart1, refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

            % IF ADC Duration == Gradient Duration
            elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStopAmp2, 1, 'WAIT');

            % IF ADC Duration < Gradient Duration
            elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.adc(i).duration -
refDur) > tol % IF RF duration < gradient duration
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
```

```
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(SequenceLoop.(char(channel2))(i).flatTime -
SequenceLoop.adc(i).duration));
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop2,
SequenceLoop.(char(channel2grad))(i).duration, gradInputStopAmp2, 1, 'WAIT');

              end
            end

        elseif (adcPresent == 0) && (rfPresent == 0) % No RF or ADC
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel2grad))(i).flatTime);
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop2,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp2, 1, 'WAIT');

        % Write Ramps Down
        elseif (adcPresent == 0) && (rfPresent == 1)
            % FLAG: Add Conditional to RF Turn off
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
                % Do nothing Case
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
        end
    end
%% Arbitrary Case
    if (SequenceLoop.(char(channel1grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "grad")
    % Two Separate Channels for Starts due to differences in shapes
    % Building Start Gradient Input because all events in a block are assumed to start
        % at the same time
        if channel1grad == "gx"
            gradInputStart1 = strjoin({gradInputStart1,'"', string(SequenceLoop.gx(i).arbpatname),'",'});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
        elseif channel2grad == "gx"
            gradInputStart1 = strjoin({gradInputStart1, '"', string(SequenceLoop.gx(i).arbpatname),'",'});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
        else
            gradInputStart1 = strjoin({gradInputStart1, '"",'});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
        end
```

```
    if channel1grad == "gy"
       gradInputStart1 = strjoin({gradInputStart1,'"', string(SequenceLoop.gy(i).arbpatname),'",'});
       gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp,'});
    elseif channel2grad == "gy"
       gradInputStart1 = strjoin({gradInputStart1, '"', string(SequenceLoop.gy(i).arbpatname),'",'});
       gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp, '});
    else
       gradInputStart1 = strjoin({gradInputStart1, '"",'});
       gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
    end

    if channel1grad == "gz"
       gradInputStart1 = strjoin({gradInputStart1,'"', string(SequenceLoop.gz(i).arbpatname),'",'});
       gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
    elseif channel2grad == "gz"
       gradInputStart1 = strjoin({gradInputStart1, '"', string(SequenceLoop.gz(i).arbpatname),'",'});
       gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
    else
       gradInputStart1 = strjoin({gradInputStart1, '"",'});
       gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
    end

    % Building Stop Gradient Input Separately
    % Stops not needed for arbritary shapes

    if rfPresent == 1
        % IF RF duration > Gradient Duration
        if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n',
'obl_shapedgradient',gradInputStart1, refDur, gradInputStartAmp1, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
% FLAG MAY NEED TO CHANGE TO BREAK UP SHAPE SO THAT

        % IF RF Duration == Gradient flat time
        elseif abs(SequenceLoop.rf(i).duration - refDur) < tol % IF RF duration == trap gradient flat time
          fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n',
'obl_shapedgradient',gradInputStart1, refDur, gradInputStartAmp1, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

        % IF RF Duration < Gradient Duration
        elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
% IF RF duration < gradient duration
          % IF RF pulse ends During Flat Time
          fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n',
'obl_shapedgradient',gradInputStart1, refDur, gradInputStartAmp1, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
        end
```

```
    end
    if adcPresent == 1
        % Gradient RF ADC Case
        if rfPresent == 1
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

        % Gradient ADC Case
        else
            % IF ADC Duration > Gradient Duration
            if ((SequenceLoop.adc(i).duration - refDur) > 0) && (abs(SequenceLoop.adc(i).duration - refDur)
> tol)
                fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n',
'obl_shapedgradient',gradInputStart1, SequenceLoop.(char(channel1grad))(i).duration,
gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

            % IF ADC Duration == Gradient Duration
            elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
                fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n',
'obl_shapedgradient',gradInputStart1, SequenceLoop.(char(channel1grad))(i).duration,
gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');


            % IF ADC Duration < Gradient Duration
            elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.adc(i).duration -
refDur) > tol % IF RF duration < gradient duration
                fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n',
'obl_shapedgradient',gradInputStart1, SequenceLoop.(char(channel1grad))(i).duration,
gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

            end
        end

    elseif (adcPresent == 0) && (rfPresent == 0) % No RF or ADC
        fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1, 'WAIT');
    end
```

```
    end
 %% Zero all the gradients
 fprintf(fileID, '\t\t%1$s(%2$s);\n', 'zero_all_gradients', '');
end
```

threeGradients.m

```
function [  ] = threeGradients_sameDurGradMod( SequenceLoop,i,fileID, input1, input2, input3, rfPresent,
adcPresent,tol )
%% Declarations
gradInputStart1 = '';
gradInputStart2 = '';
gradInputStartAmp1 = '';
gradInputStartAmp2 = '';
gradInputStop1 = '';
gradInputStopAmp1 = '';
%% Determine the channel with the longer gradient
   % This is made in an effort to streamline code cases.
   % Need to verify this is working properly, may need to add the
   % tolerance for floating point numbers
   % Note that I've kept the input 1,2,3, primarily to keep the
   % consistency between functions.  Could be modified to be hardcoded as
   % X,Y,Z
    if SequenceLoop.(char(input1))(i).gradtype == "trap"
       compVal1 = SequenceLoop.(char(input1))(i).flatTime;
    elseif SequenceLoop.(char(input1))(i).gradtype == "grad"
       compVal1 = SequenceLoop.(char(input1))(i).duration;
    end

    if SequenceLoop.(char(input2))(i).gradtype == "trap"
       compVal2 = SequenceLoop.(char(input2))(i).flatTime;
    elseif SequenceLoop.(char(input2))(i).gradtype == "grad"
       compVal2 = SequenceLoop.(char(input2))(i).duration;
    end

    if SequenceLoop.(char(input3))(i).gradtype == "trap"
       compVal3 = SequenceLoop.(char(input3))(i).flatTime;
    elseif SequenceLoop.(char(input3))(i).gradtype == "grad"
       compVal3 = SequenceLoop.(char(input3))(i).duration;
    end
%% Order
    if compVal1 >= compVal2
       if  compVal1 > compVal3
          % 1 2 3
          if compVal2 >= compVal3
            channel1grad = input1;
            channel2grad = input2;
            channel3grad = input3;
          % 1 3 2
          elseif compVal3 > compVal2
            channel1grad = input1;
            channel2grad = input3;
            channel3grad = input2;
          end
       elseif compVal3 >= compVal1
          % 3 1 2
          channel1grad = input3;
```

```
            channel2grad = input1;
            channel3grad = input2;
          end
        elseif compVal2 > compVal1
          if  compVal2 > compVal3
            % 2 1 3
            if compVal1 >= compVal3
              channel1grad = input2;
              channel2grad = input1;
              channel3grad = input3;
            % 2 3 1
            elseif compVal3 > compVal1
              channel1grad = input2;
              channel2grad = input3;
              channel3grad = input1;
            end
          elseif compVal3 >= compVal2
            % 3 2 1
            channel1grad = input3;
            channel2grad = input2;
            channel3grad = input1;
          end
        end
        % Set reference duration for all gradient events as equivalent
        if (SequenceLoop.(char(channel1grad))(i).gradtype == "trap")
          refDur = SequenceLoop.(char(channel1grad))(i).flatTime;
        elseif (SequenceLoop.(char(channel1grad))(i).gradtype == "grad")
          refDur = SequenceLoop.(char(channel1grad))(i).duration;
        end
    %% Cases
    %% TRAP TRAP TRAP
    if (SequenceLoop.(char(channel1grad))(i).gradtype == "trap") &&
    (SequenceLoop.(char(channel2grad))(i).gradtype == "trap") &&
    (SequenceLoop.(char(channel3grad))(i).gradtype == "trap")
     % Define Starts: One for Traps
            if channel1grad == "gx"
              gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
              gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
            elseif channel2grad == "gx"
              gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
              gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ',',});
            elseif channel3grad == "gx"
              gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
              gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ',',});
            else
              gradInputStart1 = strjoin({gradInputStart1, '"",'});
              gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
            end
            if channel1grad == "gy"
              gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
              gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
            elseif channel2grad == "gy"
              gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
```

```
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
  elseif channel3grad == "gy"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ',',});
  else
    gradInputStart1 = strjoin({gradInputStart1, '"",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
  end
  if channel1grad == "gz"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
  elseif channel2grad == "gz"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
  elseif channel3grad == "gz"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ',',});
  else
    gradInputStart1 = strjoin({gradInputStart1, '"",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
  end
% Define Stops: One for Traps
  if channel1grad == "gx"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  elseif channel2grad == "gx"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  elseif channel3grad == "gx"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  else
    gradInputStop1 = strjoin({gradInputStop1, '"",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
  end
  if channel1grad == "gy"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
  elseif channel2grad == "gy"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
  elseif channel3grad == "gy"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
  else
    gradInputStop1 = strjoin({gradInputStop1, '"",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
  end
  if channel1grad == "gz"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
  elseif channel2grad == "gz"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
```

```
            gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
          elseif channel3grad == "gz"
            gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
            gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
          else
            gradInputStop1 = strjoin({gradInputStop1, '"",'});
            gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
          end

        % IF RF is present, write gradients and RF Command
          if rfPresent == 1
          % Compare RF Duration to Gradient Flat Time
            % RF > Gradient Duration, custom trap gradient file needed
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
              fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
              fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
              fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).riseTime);
              fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

            % RF == Gradient Duration
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
              fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

            % RF  < Gradient Duration, need additional delay
            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
              fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
            end
          end
        % Determine if ADC is present
          if adcPresent == 1
          % If RF present, add ADC Command
            if rfPresent == 1
              % Add ADC Command
              fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
              fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
              fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

            % Ramp down Trap gradients if ADC Present
```

```
            % RF > Gradient Duration, no ramp down needed
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
               % Do nothing case, ramp downs included in the shape file

            % RF == Gradient Duration, just ramp down
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
               fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            % RF  < Gradient Duration, need additional delay, ramp downs
            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
               fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
               fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            end
         % Otherwise, write gradients and ADC event
           else
            % Compare ADC Duration to Gradient Flat Time
               % ADC > Gradient Duration, custom trap gradient file needed
               if ((SequenceLoop.adc(i).duration - refDur) > 0) && (abs(SequenceLoop.adc(i).duration - refDur)
> tol)
                  warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
                  fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after
a specific duration.*/\n');
                  fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom
shape file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
                  fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1))(i).riseTime);
                  fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                  fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                  fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

               % ADC == Gradient Duration
               elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
                  fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
                  fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                  fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                  fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                  fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

               % ADC  < Gradient Duration, need additional delay
               elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.adc(i).duration -
refDur) > tol % IF RF duration < gradient duration
                  fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
                  fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
```

```
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(SequenceLoop.(char(channel1))(i).flatTime -
SequenceLoop.adc(i).duration));
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
          end
        elseif (rfPresent == 1) && (adcPresent == 0)
          % Ramp down Trap gradients if no ADC Present
            % RF > Gradient Duration, no ramp down needed
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              % Do nothing Case
            % RF == Gradient Duration, just ramp down
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            % RF  < Gradient Duration, need additional delay,
            % ramp downs
            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
              fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end

        elseif (rfPresent == 0) && (adcPresent == 0)
          % Print Out Gradients without other events
          % Ramp ups, delay == flatTime, ramp downs
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur);
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
        end
end
%% TRAP TRAP GRAD
if (SequenceLoop.(char(channel1grad))(i).gradtype == "trap") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "trap") &&
(SequenceLoop.(char(channel3grad))(i).gradtype == "grad")
 % Define Starts: One for Traps, One for Grads:
  if channel1grad == "gx"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  elseif channel2grad == "gx"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  else
    gradInputStart1 = strjoin({gradInputStart1, '"",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
  end
  if channel3grad == "gx"
```

```
   gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gx(i).arbpatname});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gy"
   gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
elseif channel2grad == "gy"
   gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel3grad == "gy"
   gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gy(i).arbpatname});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp', ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
elseif channel2grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel3grad == "gz"
   gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gz(i).arbpatname});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

% Define Stops: One for Traps, None for Grads
if channel1grad == "gx"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
elseif channel2grad == "gx"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
```

```
   end
   if channel1grad == "gy"
      gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
   elseif channel2grad == "gy"
      gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
   else
      gradInputStop1 = strjoin({gradInputStop1, '"",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
   end
   if channel1grad == "gz"
      gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
   elseif channel2grad == "gz"
      gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
   else
      gradInputStop1 = strjoin({gradInputStop1, '"",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
   end

    % IF RF is present, write gradients and RF Command
     if rfPresent == 1
        % Compare RF Duration to Gradient Flat Time
          % RF > Gradient Duration, custom trap gradient file needed
          if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
            fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
            fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).riseTime);
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n',
'obl_shapedgradient',gradInputStart2,refDur, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

          % RF == Gradient Duration
          elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

          % RF  < Gradient Duration, need additional delay
          elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
```

```
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
          end
      end
    % Determine if ADC is present
     if adcPresent == 1
      % If RF present, add ADC Command
        if rfPresent == 1
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
          fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
          % Ramps Down
           if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
           % Do nothing case, ramp downs included in the shape file

           % RF == Gradient Duration, just ramp down
           elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
             fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

           % RF  < Gradient Duration, need additional delay, ramp downs
           elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
             fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
             fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          end
        else
        % Compare ADC Duration to Gradient Flat Time
          % ADC > Gradient Duration, custom trap gradient file needed
          if (SequenceLoop.adc(i).duration - refDur > 0) && abs(SequenceLoop.adc(i).duration - refDur) > tol
            warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
            fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
            fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).riseTime);
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

          % ADC == Gradient Duration
          elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
```

```
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

          % ADC  < Gradient Duration, need additional delay
          elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.adc(i).duration - refDur) >
tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

          end
        end
    elseif (rfPresent == 1) && (adcPresent == 0)
      % Ramps Down
          if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          % Do nothing case, ramp downs included in the shape file

          % RF == Gradient Duration, just ramp down
          elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

          % RF  < Gradient Duration, need additional delay, ramp downs
          elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          end
    elseif (rfPresent == 0) && (adcPresent == 0)
      % Print Out Gradients without other events
      fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
      fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'WAIT');
      fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
```

```
    end
end
%% TRAP GRAD TRAP
if (SequenceLoop.(char(channel1grad))(i).gradtype == "trap") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel3grad))(i).gradtype == "trap")
  % Define Starts: One for Traps, One for Grads:
  if channel1grad == "gx"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  elseif channel3grad == "gx"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  else
    gradInputStart1 = strjoin({gradInputStart1, '"",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
  end
  if channel2grad == "gx"
    gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gx(i).arbpatname});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
  else
    gradInputStart2 = strjoin({gradInputStart2, '"",'});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
  end

  if channel1grad == "gy"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
  elseif channel3grad == "gy"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
  else
    gradInputStart1 = strjoin({gradInputStart1, '"",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
  end
  if channel2grad == "gy"
    gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gy(i).arbpatname});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp', ','});
  else
    gradInputStart2 = strjoin({gradInputStart2, '"",'});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
  end

  if channel1grad == "gz"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
  elseif channel3grad == "gz"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
  else
    gradInputStart1 = strjoin({gradInputStart1, '"",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
  end
```

```
  if channel2grad == "gz"
    gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gz(i).arbpatname});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude), ','});
  else
    gradInputStart2 = strjoin({gradInputStart2, '"",'});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
  end

  % Define Stops: One for Traps, None for Grads
  if channel1grad == "gx"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  elseif channel3grad == "gx"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  else
    gradInputStop1 = strjoin({gradInputStop1, '"",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
  end
  if channel1grad == "gy"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
  elseif channel3grad == "gy"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
  else
    gradInputStop1 = strjoin({gradInputStop1, '"",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
  end
  if channel1grad == "gz"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
  elseif channel3grad == "gz"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
  else
    gradInputStop1 = strjoin({gradInputStop1, '"",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
  end

   % IF RF is present, write gradients and RF Command
    if rfPresent == 1
    % Compare RF Duration to Gradient Flat Time
      % RF > Gradient Duration, custom trap gradient file needed
      if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
        warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
        fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
        fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
        fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).riseTime);
```

```
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

        % RF == Gradient Duration
        elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

        % RF  < Gradient Duration, need additional delay
        elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
        end
      end
    % Determine if ADC is present
      if adcPresent == 1
      % If RF present, add ADC Command
        if rfPresent == 1
          % Add ADC Command
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
          fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
          % Write Ramp Downs
          % RF > Gradient Duration, custom trap gradient file needed
          if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
             % Do Nothing Case

          % RF == Gradient Duration
          elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

          % RF  < Gradient Duration, need additional delay
          elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          end
      % Otherwise, write gradients and ADC event
```

```
        else
        % Compare ADC Duration to Gradient Flat Time
          % ADC > Gradient Duration, custom trap gradient file needed
          if (SequenceLoop.adc(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
            fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
            fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).riseTime);
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
          % ADC == Gradient Duration
          elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          % ADC  < Gradient Duration, need additional delay
          elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          end
        end
      elseif (rfPresent == 1) && (adcPresent == 0)
        % Ramp down Trap gradients if no ADC Present
          % RF > Gradient Duration, custom trap gradient file needed
          if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
             % Do Nothing Case

          % RF == Gradient Duration
          elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
```

```
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

          % RF  < Gradient Duration, need additional delay
          elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',(refDur - SequenceLoop.rf(i).duration));
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          end
      elseif (rfPresent == 0) && (adcPresent == 0)
        % Print Out Gradients without other events
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'WAIT');
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4$s%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

      end
end
%% TRAP GRAD GRAD
if (SequenceLoop.(char(channel1grad))(i).gradtype == "trap") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel3grad))(i).gradtype == "grad")
  % Define Starts: One for Trap, One for Grads (bc same duration):
  if channel1grad == "gx"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  else
    gradInputStart1 = strjoin({gradInputStart1, '"",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
  end
  if channel2grad == "gx"
    gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gx(i).arbpatname});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
  elseif channel3grad == "gx"
    gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gx(i).arbpatname});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
  else
    gradInputStart2 = strjoin({gradInputStart2, '"",'});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
  end

  if channel1grad == "gy"
    gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
  else
    gradInputStart1 = strjoin({gradInputStart1, '"",'});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
  end
  if channel2grad == "gy"
    gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gy(i).arbpatname});
    gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp', ','});
```

```
elseif channel3grad == "gy"
   gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gy(i).arbpatname});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp', ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1, '"ramp_hold",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel2grad == "gz"
   gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gz(i).arbpatname});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude), ','});
elseif channel3grad == "gz"
   gradInputStart2 = strjoin({gradInputStart2, SequenceLoop.gz(i).arbpatname});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

% Define Stops: One for Trap, None for Grads:
if channel1grad == "gx"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
end
if channel1grad == "gy"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
end
if channel1grad == "gz"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
end

% IF RF is present, write gradients and RF Command
   if rfPresent == 1
   % Compare RF Duration to Gradient Flat Time
      % RF > Gradient Duration
```

```
        if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
          fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
          fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).riseTime);
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

        % RF == Gradient Duration
        elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

        % RF  < Gradient Duration, need additional delay
        elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));

        end
      end
  % Determine if ADC is present
      if adcPresent == 1
      % If RF present, add ADC Command
        if rfPresent == 1
          % Add ADC Command
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
          fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
          % RF > Gradient Duration
          if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            % Do nothing case
          % RF == Gradient Duration
          elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
```

```
            % RF  < Gradient Duration, need additional delay
            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
               fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.rf(i).duration);
               fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
         % Otherwise, write gradients and ADC event
           else
            % Compare ADC Duration to Gradient Flat Time
              % ADC > Gradient Duration
              if (SequenceLoop.adc(i).duration - refDur > 0) && abs(SequenceLoop.adc(i).duration - refDur) > tol
                warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
                fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
                fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1,
'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).riseTime);
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');

              % ADC == Gradient Duration
              elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

              % ADC  < Gradient Duration, need additional delay
              elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.adc(i).duration - refDur) >
tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
```

```
            end
          end
      elseif (rfPresent == 1) && (adcPresent == 0)
          % Ramp down Trap gradients if no ADC Present
            % RF > Gradient Duration
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              % Do nothing case
            % RF == Gradient Duration
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');

            % RF  < Gradient Duration, need additional delay
            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.rf(i).duration);
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
      elseif (rfPresent == 0) && (adcPresent == 0)
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).riseTime, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
refDur, gradInputStartAmp2, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel1grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
      end
end
%% GRAD TRAP TRAP
if (SequenceLoop.(char(channel1grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "trap") &&
(SequenceLoop.(char(channel3grad))(i).gradtype == "trap")
% Define Starts: One for Traps (bc same duration), One for Grad :
   if channel1grad == "gx"
      gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gx(i).arbpatname});
      gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
   else
      gradInputStart1 = strjoin({gradInputStart1, '"",'});
      gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
   end
   if channel2grad == "gx"
      gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
      gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
   elseif channel3grad == "gx"
      gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
      gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
   else
      gradInputStart2 = strjoin({gradInputStart2, '"",'});
      gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
   end

   if channel1grad == "gy"
      gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gy(i).arbpatname});
      gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
```

```
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel2grad == "gy"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp', ','});
elseif channel3grad == "gy"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp', ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gz(i).arbpatname});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel2grad == "gz"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude), ','});
elseif channel3grad == "gz"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

% Define Stops: One for Traps, None for Grads
if channel2grad == "gx"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
elseif channel3grad == "gx"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
end
if channel2grad == "gy"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
elseif channel3grad == "gy"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
```

```
    end
    if channel2grad == "gz"
        gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
        gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
    elseif channel3grad == "gz"
        gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
        gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
    else
        gradInputStop1 = strjoin({gradInputStop1, '"",'});
        gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
    end

     % IF RF is present, write gradients and RF Command
        if rfPresent == 1
        % Compare RF Duration to Gradient Flat Time
            % RF > Gradient Duration, custom trap gradient file needed
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
                warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
                fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
                fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1,
'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel2grad))(i).riseTime);
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
            % RF == Gradient Duration
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
            % RF  < Gradient Duration, need additional delay
            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
            end

        end
    % Determine if ADC is present
        if adcPresent == 1
        % If RF present, add ADC Command
```

```
        if rfPresent == 1
            % Add ADC Command
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            % Write Ramp Downs
            % RF > Gradient Duration, custom trap gradient file needed
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
                % Do Nothing Case
            % RF == Gradient Duration
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            % RF  < Gradient Duration, need additional delay
            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
        % Otherwise, write gradients and ADC event
            else
            % Compare ADC Duration to Gradient Flat Time
            % ADC > Gradient Duration, custom trap gradient file needed
            if (SequenceLoop.adc(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
                warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
                fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
                fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1,
'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel2grad))(i).riseTime);
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            % ADC == Gradient Duration
            elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            % ADC  < Gradient Duration, need additional delay
            elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
```

```
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          end
        end
      elseif (rfPresent == 1) && (adcPresent == 0)
        % Ramp down Trap gradients if no ADC Present
        % RF > Gradient Duration, custom trap gradient file needed
        if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          % Do Nothing Case
        % RF == Gradient Duration
        elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
        % RF  < Gradient Duration, need additional delay
        elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
        end
      elseif (rfPresent == 0) && (adcPresent == 0)
        % Print Out Gradients without other events
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'WAIT');
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
      end
end
%% GRAD TRAP GRAD
if (SequenceLoop.(char(channel1grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "trap") &&
(SequenceLoop.(char(channel3grad))(i).gradtype == "grad")
% Define Starts: One for Trap, One for Grads (bc same duration):
  if channel1grad == "gx"
    gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gx(i).arbpatname});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  elseif channel3grad == "gx"
    gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gx(i).arbpatname});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
  else
    gradInputStart1 = strjoin({gradInputStart1, '""',});
    gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
  end
```

```
if channel2grad == "gx"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gy"
   gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gy(i).arbpatname});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
elseif channel3grad == "gy"
   gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gy(i).arbpatname});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel2grad == "gy"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp', ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gz(i).arbpatname});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
elseif channel3grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gz(i).arbpatname});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel2grad == "gz"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

% Define Stops: One for Trap, None for Grads:
if channel2grad == "gx"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
end
if channel2grad == "gy"
```

```
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
  else
    gradInputStop1 = strjoin({gradInputStop1, '"",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
  end
  if channel2grad == "gz"
    gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
  else
    gradInputStop1 = strjoin({gradInputStop1, '"",'});
    gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
  end

   % IF RF is present, write gradients and RF Command
    if rfPresent == 1
    % Compare RF Duration to Gradient Flat Time
      % RF > Gradient Duration, custom trap gradient file needed
      if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
        fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
        fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1,
'NOWAIT');
        fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel2grad))(i).riseTime);
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
        fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
      % RF == Gradient Duration
      elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
        fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
      % RF  < Gradient Duration, need additional delay
      elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
        fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
      end
    end
   % Determine if ADC is present
    if adcPresent == 1
```

```
    % If RF present, add ADC Command
      if rfPresent == 1
        % Add ADC Command
        fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
        fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
        fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
        % Write Ramp Downs
        % RF > Gradient Duration, custom trap gradient file needed
        if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          % Do nothing case
        % RF == Gradient Duration
        elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
        % RF  < Gradient Duration, need additional delay
        elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.rf(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
        end
    % Otherwise, write gradients and ADC event
      else
      % Compare ADC Duration to Gradient Flat Time
        % ADC > Gradient Duration, custom trap gradient file needed
        if (SequenceLoop.adc(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
          warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
          fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
          fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel2grad))(i).duration, gradInputStartAmp2, 1,
'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel2grad))(i).riseTime);
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
          fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
        % ADC == Gradient Duration
        elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
          fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
        % ADC  < Gradient Duration, need additional delay
```

```
            elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
          end
        elseif (rfPresent == 1) && (adcPresent == 0)
          % Ramp down Trap gradients if no ADC Present
            % RF > Gradient Duration, custom trap gradient file needed
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              % Do nothing case
            % RF == Gradient Duration
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            % RF  < Gradient Duration, need additional delay
            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.rf(i).duration);
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
        elseif (rfPresent == 0) && (adcPresent == 0)
          % Print Out Gradients without other events
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel2grad))(i).riseTime, gradInputStartAmp2, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel2grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
        end
end
%% GRAD GRAD TRAP
if (SequenceLoop.(char(channel1grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel3grad))(i).gradtype == "trap")
  % Define Starts: One for Trap, One for Grads (bc same duration):
    if channel1grad == "gx"
      gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gx(i).arbpatname});
      gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
    elseif channel2grad == "gx"
      gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gx(i).arbpatname});
      gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
    else
      gradInputStart1 = strjoin({gradInputStart1, '""',});
```

```
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel3grad == "gx"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gy"
   gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gy(i).arbpatname});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
elseif channel2grad == "gy"
   gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gy(i).arbpatname});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel3grad == "gy"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, 'gpeAmp', ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

if channel1grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gz(i).arbpatname});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
elseif channel2grad == "gz"
   gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gz(i).arbpatname});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart1 = strjoin({gradInputStart1, '"",'});
   gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
end
if channel3grad == "gz"
   gradInputStart2 = strjoin({gradInputStart2, '"ramp_hold",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, num2str(SequenceLoop.gz(i).amplitude), ','});
else
   gradInputStart2 = strjoin({gradInputStart2, '"",'});
   gradInputStartAmp2 = strjoin({gradInputStartAmp2, '0,'});
end

% Define Stops: One for Trap, None for Grads:
if channel3grad == "gx"
   gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
else
   gradInputStop1 = strjoin({gradInputStop1, '"",'});
   gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
```

```
   end
   if channel3grad == "gy"
      gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, 'gpeAmp', ','});
   else
      gradInputStop1 = strjoin({gradInputStop1, '"",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
   end
   if channel3grad == "gz"
      gradInputStop1 = strjoin({gradInputStop1, '"ramp_down",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
   else
      gradInputStop1 = strjoin({gradInputStop1, '"",'});
      gradInputStopAmp1 = strjoin({gradInputStopAmp1, '0,'});
   end

    % IF RF is present, write gradients and RF Command
      if rfPresent == 1
      % Compare RF Duration to Gradient Flat Time
         % RF > Gradient Duration, custom trap gradient file needed
         if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
            fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
            fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel3grad))(i).duration, gradInputStartAmp2, 1,
'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel3grad))(i).riseTime);
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
         % RF == Gradient Duration
         elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel3grad))(i).rampTime, gradInputStartAmp2, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
         % RF  < Gradient Duration, need additional delay
         elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel3grad))(i).rampTime, gradInputStartAmp2, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
         end
      end
```

```
    % Determine if ADC is present
      if adcPresent == 1
      % If RF present, add ADC Command
        if rfPresent == 1
          % Add ADC Command
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
          fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
          fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
          % Write Ramp Downs
          % RF > Gradient Duration, custom trap gradient file needed
          if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            % Do Nothing Case
          % RF == Gradient Duration
          elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel3grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          % RF  < Gradient Duration, need additional delay
          elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.rf(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel3grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
          end
      % Otherwise, write gradients and ADC event
        else
        % Compare ADC Duration to Gradient Flat Time
          % ADC > Gradient Duration, custom trap gradient file needed
          if (SequenceLoop.adc(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
            warndlg('See Commented Lines in File' ,'Warning: Out File Needs Custom Edit');
            fprintf(fileID,'\t\t/*User needs to input custom trapezoidal shape names that ramp down after a
specific duration.*/\n');
            fprintf(fileID, '\t\t/*%1$s(%2$s%3$f,%4s,%5$d,%6$s);*/\n', 'obl_shapedgradient','custom shape
file names: gro, gpe, gss', SequenceLoop.(char(channel3grad))(i).duration, gradInputStartAmp2, 1,
'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel3grad))(i).riseTime);
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
          % ADC == Gradient Duration
          elseif abs(SequenceLoop.adc(i).duration - refDur) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel3grad))(i).rampTime, gradInputStartAmp2, 1, 'WAIT');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel3grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
```

```
            % ADC  < Gradient Duration, need additional delay
            elseif (SequenceLoop.adc(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) >
tol
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel3grad))(i).rampTime, gradInputStartAmp2, 1, 'WAIT');
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'NOWAIT');
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
                fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.adc(i).duration);
                fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel3grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
          end
        elseif (rfPresent == 1) && (adcPresent == 0)
          % Ramp down Trap gradients if no ADC Present
            % RF > Gradient Duration, custom trap gradient file needed
            if (SequenceLoop.rf(i).duration - refDur > 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              % Do Nothing Case
            % RF == Gradient Duration
            elseif abs(SequenceLoop.rf(i).duration - refDur) < tol
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel3grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            % RF  < Gradient Duration, need additional delay
            elseif (SequenceLoop.rf(i).duration - refDur < 0) && abs(SequenceLoop.rf(i).duration - refDur) > tol
              fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',refDur - SequenceLoop.rf(i).duration);
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel3grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
            end
        elseif (rfPresent == 0) && (adcPresent == 0)
          % Print Out Gradients without other events
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart2,
SequenceLoop.(char(channel3grad))(i).rampTime, gradInputStartAmp2, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
refDur, gradInputStartAmp1, 1, 'WAIT');
          fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStop1,
SequenceLoop.(char(channel3grad))(i).fallTime, gradInputStopAmp1, 1, 'WAIT');
        end

end
%% GRAD GRAD GRAD
if (SequenceLoop.(char(channel1grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel2grad))(i).gradtype == "grad") &&
(SequenceLoop.(char(channel3grad))(i).gradtype == "grad")
 % Define Starts: One for Traps
          if channel1grad == "gx"
            gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gx(i).arbpatname});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ','});
          elseif channel2grad == "gx"
            gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gx(i).arbpatname});
```

```
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ',',});
        elseif channel3grad == "gx"
            gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gx(i).arbpatname});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gx(i).amplitude), ',',});
        else
            gradInputStart1 = strjoin({gradInputStart1, '""','});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
        end
        if channel1grad == "gy"
            gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gy(i).arbpatname});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
        elseif channel2grad == "gy"
            gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gy(i).arbpatname});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ','});
        elseif channel3grad == "gy"
            gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gy(i).arbpatname});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, 'gpeAmp', ',',});
        else
            gradInputStart1 = strjoin({gradInputStart1, '""','});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
        end
        if channel1grad == "gz"
            gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gz(i).arbpatname});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
        elseif channel2grad == "gz"
            gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gz(i).arbpatname});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ','});
        elseif channel3grad == "gz"
            gradInputStart1 = strjoin({gradInputStart1, SequenceLoop.gz(i).arbpatname});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, num2str(SequenceLoop.gz(i).amplitude), ',',});
        else
            gradInputStart1 = strjoin({gradInputStart1, '""','});
            gradInputStartAmp1 = strjoin({gradInputStartAmp1, '0,'});
        end
      % Define Stops: None for Grads

    % IF RF is present, write gradients and RF Command
      if rfPresent == 1
      % Compare RF Duration to Gradient Flat Time
        % RF > Gradient Duration, custom trap gradient file needed
        if (SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration > 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) > tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
        % RF == Gradient Duration
        elseif abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) < tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1, 'NOWAIT');
```

```
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
          % RF  < Gradient Duration, need additional delay
          elseif (SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration < 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) > tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s,%3$s,%4$s,%5$s,%6$s);\n', 'shaped_pulse',
string(SequenceLoop.rf(i).pulsepatname), SequenceLoop.rf(i).duration,
string(SequenceLoop.rf(i).phaseVarName),string(SequenceLoop.rf(i).RG1), string(SequenceLoop.rf(i).RG2));
          end
        end
      % Determine if ADC is present
        if adcPresent == 1
        % If RF present, add ADC Command
          if rfPresent == 1
            % Add ADC Command
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            % Write Ramp Downs
            % RF > Gradient Duration, custom trap gradient file needed
              if (SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration > 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) > tol
                % Do Nothing
              % RF == Gradient Duration
              elseif abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) < tol
                % Do Nothing
              % RF  < Gradient Duration, need additional delay
              elseif (SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration < 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) > tol
                fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).duration -
SequenceLoop.rf(i).duration);
              end
        % Otherwise, write gradients and ADC event
          else
          % Compare ADC Duration to Gradient Flat Time
            % ADC > Gradient Duration, custom trap gradient file needed
            if (SequenceLoop.adc(i).duration - SequenceLoop.(char(channel1grad))(i).duration > 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) > tol
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1, 'NOWAIT');
              fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
              fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
              fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            % ADC == Gradient Duration
            elseif abs(SequenceLoop.adc(i).duration - SequenceLoop.(char(channel1grad))(i).duration) < tol
              fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1, 'NOWAIT');
              fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
```

```
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
          % ADC  < Gradient Duration, need additional delay
          elseif (SequenceLoop.adc(i).duration - SequenceLoop.(char(channel1grad))(i).duration < 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) > tol
            fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1, 'NOWAIT');
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2on','');
            fprintf(fileID, '\t\t%1$s(%2$d,%3$f);\n', 'acquire', SequenceLoop.adc(i).N,
SequenceLoop.adc(i).duration);
            fprintf(fileID, '\t\t%1$s(%2$s);\n', 'sp2off','');
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).duration -
SequenceLoop.adc(i).duration);
          end
        end
    elseif (rfPresent == 1) && (adcPresent == 0)
        % Ramp down Trap gradients if no ADC Present
          % RF > Gradient Duration, custom trap gradient file needed
          if (SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration > 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) > tol
            % Do Nothing
          % RF == Gradient Duration
          elseif abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) < tol
            % Do Nothing
          % RF  < Gradient Duration, need additional delay
          elseif (SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration < 0) &&
abs(SequenceLoop.rf(i).duration - SequenceLoop.(char(channel1grad))(i).duration) > tol
            fprintf(fileID, '\t\t%1$s(%2$f);\n', 'delay',SequenceLoop.(char(channel1grad))(i).duration -
SequenceLoop.rf(i).duration);
          end
    elseif (rfPresent == 0) && (adcPresent == 0)
        % Print Out Gradients without other events
        fprintf(fileID, '\t\t%1$s(%2$s%3$f,%4s,%5$d,%6$s);\n', 'obl_shapedgradient',gradInputStart1,
SequenceLoop.(char(channel1grad))(i).duration, gradInputStartAmp1, 1, 'WAIT');
    end

end
%% Zero all the gradients
 fprintf(fileID, '\t\t%1$s(%2$s);\n', 'zero_all_gradients', '');
end
```

APPENDIX B

Development of Supporting Functions

Note: Contained within this section is the supplemental functions included for ease of use with integration at the MRSL.

makeGaussPulse.m

```
function [rf, gz] = makeGaussPulse(flip,varargin)
%makeSincPulse Create a slice selective gaussian pulse.
%   rf=makeGaussPulse(flip, 'Duration', dur) Create gaussian pulse
%   with given flip angle (rad) and duration (s).
%
%   rf=makeGaussPulse(..., 'FreqOffset', f,'PhaseOffset',p)
%   Create gauss pulse with frequency offset (Hz) and phase offset (rad).
%
%   [rf, gz]=makeGaussPulse(...,'SliceThickness',st) Return the
%   slice select gradient corresponding to given slice thickness (m).
%
%   [rf, gz]=makeGaussPulse(flip,lims,...) Create slice selection gradient
%   with the specificed gradient limits (e.g. amplitude, slew).
%
%   See also  Sequence.addBlock

persistent parser
if isempty(parser)
   parser = inputParser;
   parser.FunctionName = 'makeSincPulse';

   % RF params
   addRequired(parser,'flipAngle',@isnumeric);
   addOptional(parser,'system',mr.opts(),@isstruct);
   addParamValue(parser,'duration',0,@isnumeric);
   addParamValue(parser,'freqOffset',0,@isnumeric);
   addParamValue(parser,'phaseOffset',0,@isnumeric);
   addParamValue(parser,'timeBwProduct',4,@isnumeric);
   addParamValue(parser,'apodization',0,@isnumeric);
   % Slice params
   addParamValue(parser,'maxGrad',0,@isnumeric);
   addParamValue(parser,'maxSlew',0,@isnumeric);
   addParamValue(parser,'sliceThickness',0,@isnumeric);
end
parse(parser,flip,varargin{:});
opt = parser.Results;

BW=opt.timeBwProduct/opt.duration;
alpha=opt.apodization;
N=round(opt.duration/1e-6);
t = (1:N)*opt.system.rfRasterTime;
tt = t - opt.duration/2;
```

```
window = (1.0-alpha+alpha*cos(2*pi*tt/opt.duration));
signal = window.*gauss(BW*tt);
flip=sum(signal)*opt.system.rfRasterTime*2*pi;
signal=signal*opt.flipAngle/flip;

rf.type = 'rf';
rf.signal = signal;
rf.t = t;
rf.freqOffset = opt.freqOffset;
rf.phaseOffset = opt.phaseOffset;
rf.deadTime = opt.system.rfDeadTime;
rf.ringdownTime = opt.system.rfRingdownTime;

fillTime=0;
if nargout>1
   assert(opt.sliceThickness>0,'SliceThickness must be provided');
   if opt.maxGrad>0
      opt.system.maxGrad = opt.maxGrad;
   end
   if opt.maxSlew>0
      opt.system.maxSlew = opt.maxSlew;
   end

   amplitude = BW/opt.sliceThickness;
   area = amplitude*opt.duration;
   gz = mr.makeTrapezoid('z',opt.system,'flatTime',opt.duration,'flatArea',area);

   % Pad RF pulse with zeros during gradient ramp up
   fillTime=gz.riseTime;
   tFill = (1:round(fillTime/1e-6))*1e-6;   % Round to microsecond
   rf.t = [tFill rf.t+tFill(end) ];
   rf.signal=[zeros(size(tFill)), rf.signal];
end

% Add dead time to start of pulse, if required
if fillTime<rf.deadTime
   fillTime=rf.deadTime-fillTime;
   tFill = (1:round(fillTime/1e-6))*1e-6;   % Round to microsecond
   rf.t = [tFill rf.t+tFill(end) ];
   rf.signal=[zeros(size(tFill)), rf.signal];
end

if rf.ringdownTime>0
   tFill = (1:round(rf.ringdownTime/1e-6))*1e-6;  % Round to microsecond
   rf.t = [rf.t rf.t(end)+tFill];
   rf.signal = [rf.signal, zeros(size(tFill))];
end


function y=gauss(x)
   % gauss Calculate the gauss function:
   %   gauss(x) = exp(-at^2)
   %
```

```
    % This is a useful helper function for those without the signal processing
    % toolbox

    i=find(x==0);
    x(i)= 1;
    y = exp(-1 .* x.^2);
    y(i) = 1;
end

end
```

makeHalfSine.m

```
function grad = makeHalfSine(channel, varargin)

persistent parser

if isempty(parser)
    parser = inputParser;
    parser.FunctionName = 'makeHalfSine';

    validChannels = {'x','y','z'};
    parser.addRequired('channel',...
        @(x) any(validatestring(x,validChannels)));
    parser.addOptional('system',mr.opts(),@isstruct);
    parser.addParameter('duration',0,@(x)(isnumeric(x) && x>0));
    parser.addParameter('amplitude',[],@isnumeric);
    parser.addParameter('timeBwProduct',4,@isnumeric);
    parser.addParameter('apodization',0,@isnumeric);parser.addParameter('maxGrad',0,@isnumeric);
    parser.addParameter('maxSlew',0,@isnumeric);


end
parse(parser,channel,varargin{:});
opt = parser.Results;

maxSlew=opt.system.maxSlew;
riseTime=opt.system.riseTime;
maxGrad=opt.system.maxGrad;

if opt.maxGrad>0
    maxGrad=opt.maxGrad;
end
if opt.maxSlew>0
    maxSlew=opt.maxSlew;
end


if isempty(opt.referenceTrap) && isempty(opt.duration) && isempty(opt.amplitude)
    error('makeHalfSine:invalidArguments','Must supply either ''referenceTrap'', ''duration'' or ''amplitude''');
end
```

```
BW=opt.timeBwProduct/opt.duration;
alpha=opt.apodization;
N=round(opt.duration/1e-6);
t = (1:N)*opt.system.gradRasterTime;
tt = t - opt.duration/2;
window = (1.0-alpha+alpha*cos(2*pi*tt/opt.duration));
halfsineShape = window.*sin(2*BW*tt);

grad.type = 'grad';
grad.channel = opt.channel;
grad.amplitude = amplitude;
grad.waveform = halfsineShape;
grad.t = duration;


end
```

targetMagnet.m

```
function [ system ] = targetSystem( input )
%TARGETSYSTEM Autoselect parameters particular for use with the TAMU MRSL systems
 % Currently supports the 33cm and the 40cm scanner.  Returns the targeted
 % systems necessary parameters.

 % Optional Information Includes:
   % maxGrad -- defaults to 40 mT/m
   % maxSlew -- defaults to 170 mT/m/ms
   % riseTime -- if empty, maxSlew gets set to empty
   % rfDeadTime -- 0
   % rfRingdownTime -- 0
   % adcDeadTime -- 0
   % rfRasterTime -- defaults to 1e-6
   % gradRasterTime -- defaults to 10e-6

 switch input
    case '33cm'
       system = mr.opts('MaxGrad', 100, 'GradUnit', 'mT/m', ...
          'MaxSlew', 170, 'SlewUnit', 'T/m/s', 'rfRingdownTime', 30e-6, ...
         'rfDeadTime', 100e-6, 'riseTime',  500e-6);
    case '40cm'
       system = mr.opts('MaxGrad', 30, 'GradUnit', 'mT/m', ...
          'MaxSlew', 170, 'SlewUnit', 'T/m/s', 'rfRingdownTime', 30e-6, ...
         'rfDeadTime', 100e-6, 'riseTime',  500e-6);
    otherwise
       error('System Not Recognized, Please Enter Parameters Manually using mr.opts');

end
```

APPENDIX C

User Manual

Attached are images of the user manual created as an aid to students in the MRSL.

User Guide to Using Pulseq at the MRSL

Preface

This guide serves to introduce students as to how to write pulse sequences in Pulseq for use at Texas A&M's Magnetic Resonance Research Laboratory. Within this document, there are instructions and examples to aid the user in writing pulse sequences for use within the MRSL. Presently, only the Varian 4.7T systems are supported in this document.

This document is targeted towards providing students, particularly those new to the MRSL or MR imaging, the necessary tools to write their own pulse sequences, be it as a first attempt, or part of their coursework. This document is not comprehensive to the capabilities of Pulseq, however, as it focuses on the functions to be most commonly used by the student.

It is encouraged that the students browse the functions available to them, as many are not covered in this document. Using "help *function name*" often lends a helpful explanation.

Table of Contents

Setting Up Parameters

The first step of pulse sequence writing is to define the system terms and limitations. Presently, for use at the MRSL, a function has been introduced to simplify this process.

Using the "targetMagnet" function, students new to the MRSL can define the system parameters based off the limits defined as safety limitations. The function takes a single input, in string format, either '33cm' or '40cm' designating one of the two magnets employed at the MRSL by their bore size. The example syntax is shown below:

```
system = targetMagnet('33cm');
```

Alternatively, should the user be experienced with imaging and has the proper knowledge and background to define system limitations themselves, then the user can utilize the "mr.opts" command. This takes a variety of inputs. The inputs, with their units and defaults, are listed in the table below.

Parameter Name	Description	Units	Defaulted Values
gradUnit	Units for Maximum Gradient Input	'Hz/m', 'mT/m', or 'rad/ms/mm'	'Hz/m'
slewUnit	Units for Maximum Slew Rate Input	'Hz/m/s', 'mT/m/ms', 'T/m/s' or 'rad/ms/mm/ms'	'Hz/m/s'
maxGrad	Maximum Gradient Amplitude	Defined by gradUnit	40 mT/m
maxSlew	Maximum Slew Rate	Defined by slewUnit	170 T/m/s
riseTime	Time to Ramp Up	Seconds	--
rfDeadTime	RF Dead Time	Seconds	0
rfRingdownTime	RF Ringdown Time	Seconds	0
adcDeadTime	ADC Dead Time	Seconds	0
rfRasterTime	RF Raster Time	Seconds	1E-6
gradRasterTime	Gradient Raster Time	Seconds	10E-6

It should be noted that in the event that the rise time and the slew rate are both defined, the rise time value is used and the max slew rate is set empty. Additionally, there no required inputs. Example syntax:

```
system = mr.opts('MaxGrad', 40, 'GradUnit', 'mT/m',
'riseTime', '500E-6', 'rfRingdownTime', '30E-6');
```

Additionally, a sequence object must be declared, which is what the pulse sequence will be attached to prior to using the write function to produce the '.seq' output. In order to do this,

the user employs the 'Sequence' function, which takes only the system parameters we just defined. An example is shown below:

```
seqObject = mr.Sequence(system);
```

Creating Events

RF Pulses

Definition of RF pulses has the ability to take many forms, depending on the desired shape. Introduced here are two functions that will be most commonly used within the lab, one creating a Sinc pulse, one that creates an arbitrary RF waveform to be played out. It should be noted that there are functions for a block pulse or a gaussian pulse, though those are not discussed here.

Sinc Pulses

Sinc pulses can be defined using the mr.makeSincPulse function. Here the user has several options for inputs, listed in the parameter table below:

Parameter Name	Description	Units	Defaults
flipAngle	RF Flip Angle	Radians	Required Parameter
system	Defined system	--	Calls mr.opts()
duration	Length of pulse	Seconds	0
freqOffset	Frequency Offset	Hz	0
phaseOffset	Phase Offset	Radians	0
timeBwProduct	Product of Duration and RF pulse BW	--	4
apodization	Apodization value	--	0

It should be noted here that the 'flipAngle' parameter is a required input. RF pulse definition is shown in examples below:

```
rf90 = mr.makeSincPulse(pi/2, 'Duration', 2E-3,
'timeBwProduct', 6);

rf180 = mr.makeSincPulse(pi, systemVariable, 'Duration',
2E-3, 'timeBwProduct', 6);
```

Additionally, there is the option for simultaneously defining the coinciding slice select gradient pulse. The parameters are shown in the table below.

Parameter Name	Description	Units	Defaults
maxGrad	Maximum Gradient Value	Hz/m	0
maxSlew	Max Slew Rate	'Hz/m/s'	0
sliceThickness	Slice Thickness	Meters	Required
delay	Delay before start of RF pulse	Seconds	0

From these parameters, only slice thickness is required to define a coinciding slice select gradient. An example of the simultaneous RF pulse and slice select gradient definition are shown below:

```
[rf90,  = mr.makeSincPulse(pi/2, 'Duration', 2E-3,
'timeBwProduct', 6, 'sliceThickness', 0.003);
```

Arbitrary Shapes

Given the event that the shape does not fit one of the predefined ones, the option to define an arbitrarily waveform exists in the form of the 'makeArbitraryRF' function, which receives the inputs in the table below as a means to define the desired waveform.

Parameter Name	Description	Units	Defaults
signal	Desired Complex Signal	--	Required
flipAngle	Flip Angle	Radian	Required
system	Defined system	--	mr.opts()
freqOffset	Frequency Offset	Hz	0
phaseOffset	Phase Offset	Radian	0
timeBwProduct	Time Bandwidth Product	--	0
bandwidth	RF pulse bandwidth	Hz	0

The signal and flip angle parameters are required. An example of the arbitrary RF pulse definition syntax is shown below:

```
rf90 = mr.makeArbitraryRF(pulseShape, pi/2, 'Duration', 2E-
3, 'timeBwProduct', 6);
```

Of note is that the slice select gradient can still be simultaneously defined. The additional parameters are identical to those for the Sinc pulse function, but are reproduced in the table below for convenience:

Parameter Name	Description	Units	Defaults
maxGrad	Maximum Gradient Value	Hz/m	0
maxSlew	Max Slew Rate	'Hz/m/s'	0
sliceThickness	Slice Thickness	Meters	Required
delay	Delay before start of RF pulse	Seconds	0

Again, slice thickness is the only required parameter for definition of the coinciding gradient. An example of the syntax is shown below:

```
[rf180, gz180] = mr.makeArbitraryRF(pulseShape, pi,
'Duration', 4E-3, 'timeBwProduct', 6, 'sliceThickness',
0.005);
```

Gradient Pulses

Defining gradient events allows for two main options: the commonly used trapezoidal shape, and the arbitrarily defined shape, similar to its RF counterpart. It is hoped that more functions will be added as they are recognized and developed.

Trapezoidal Gradient Events

The trapezoidal gradient is one of the more common shapes for gradient events, thus including it in Pulseq was a nice gift from the developers. The 'makeTrapezoid' function allows for a rectangular gradient event to be declared via a number of parameters. These parameters are listed in the table below:

Parameter Name	Description	Units	Defaults
channel	Channel of Gradient Event	--	Required
system	Defined system	--	mr.opts()
duration	Duration of Gradient Event	Seconds	0
area	Area under the curve of the Gradient	1/m	0
flatTime	Gradient duration not including ramps	Radian	0
flatArea	Gradient area under the curve not including ramps	1/m	0
amplitude	Gradient Amplitude	Hz/m	0
maxGrad	Maximum Gradient Value	Hz/m	0
maxSlew	Max Slew Rate	'Hz/m/s'	0
riseTime	Ramp Time for Gradient	Seconds	0

It should be noted that either there are a number of ways to declare a trapezoidal gradient event using these inputs. Two common examples are shown below;

```
gx = mr.makeTrapezoid('x', system, 'flatTime', 2E-3,
'flatArea', (gradAmplitude * 2E-3));
```

Or, rather than focus on flat times.

```
gz = mr.makeTrapezoid('z', system, 'duration', 3E-3,
'amplitude', gradAmplitude);
```

Arbitrary Gradient Events

Similarly, to the arbitrary RF function, 'makeArbitraryGrad' functions as a means to create arbitrarily shaped gradient pulses. Compared to the trapezoidal declaration, this function is relatively simple – there are two required inputs, and the system definitions. The table below shows the inputs:

Parameter Name	Description	Units	Defaults
channel	Channel of Gradient Event	X, Y or Z	Required
waveform	Desired Gradient Shape	--	Required
system	Defined system	--	mr.opts()
maxGrad	Maximum Gradient Value	Hz/m	0
maxSlew	Max Slew Rate	'Hz/m/s'	0

An example of the function is shown below:

```
gz = mr.makeArbitraryGrad('z', myWaveform, system);
```

ADC Events

ADC events are a straightforward process, with only a single function needed to define them. The 'makeAdc' function has the capabilities to accept up to 7 inputs, listed in the table below.

Parameter Name	Description	Units	Defaults
numSamples	Number of sample points	--	Required
system	Defined system	--	mr.opts()
dwell	Dwell Time	Seconds	0
duration	Acquisition Duration	Seconds	0
delay	Phase Offset	Radian	0
freqOffset	Receiver Frequency Offset	Hz	0
phaseOffset	Receiver Phase Offset	Radians	0

Of note is that either the dwell time or the duration of the ADC event must be defined, but not both. Otherwise, the function will return an error. An example of the syntax used to declare an ADC event is shown below:

```
adcEvent = mr.makeAdc(numberofPoints, 'duration', 6.4E-3);
```

Determining Timing

Calculating Durations

In order to calculate delays, it is often necessary to make calculations based off parameters. As a means to aid the user, the Pulseq developers included the 'calcDuration' function. An example of its use is shown in the example below.

```
gxTotalDuration = mr.calcDuration(trapezoidalgz90);
```

This example would result in the gxTotalDuration variable containing the entire duration of the trapezoidal shaped gz90 gradient pulse, including the ramp times.

Determining Delays

Once event durations have been determined, the calculations for delays, and the delay event themselves can be declared using the 'makeDelay' function. Depending on the type of sequence, the delays vary, but can be calculated easy using simple subtraction in Matlab. The 'makeDelay' function has a single input – the duration of the delay. An example of the syntax is shown below:

```
delay1 = mr.makeDelay(delay1duration);
```

Writing the Sequence

Sequence Construction for Looping

Once all events, including RF pulses, all gradient events, acquisition events, and delays have been declared, the base function can be written. Often, as a means to vary phase encoding gradient amplitudes, for-loops are involved. In order to construct a sequence, the base sequence is written out as a series of time blocks, each block attached to the sequence with the 'addBlock' function. The 'addBlock' function takes as many declared event inputs as is present in the block. An example of the 'addBlock syntax is shown below:

```
seqObject.addBlock(rf90, gz90);
```

Using these 'addBlock' functions for each time block, the base sequence can be defined. Putting these into a for-loop, looping the same number of times as there are phase encoding steps, will allow for the changing phase encode gradient amplitude to be accommodated, as the gradient declarations can take place within the for-loop. An example is shown below for a basic spin echo:

```
for i = 1:numberPE
     seqObject.addBlock(rf90,gz90);
     gy = mr.makeTrapezoid('y', system, 'flatTime', 2E-3,
     'flatArea', (gpeStepSize*i) * tped);
     seqObject.addBlock(gzr, gxPre, gy);
     seqObject.addBlock(delay1);
     seqObject.addBlock(rf180, gz180);
     seqObject.addBlock(delay2);
     seqObject.addBlock(gx, adc);
     seqObject.addBlock(delay3);
end
```

File Generation

Generation of the Sequence File

Now that the declared events have been attached to the seqObject object using the 'addBlock' function, the .seq file can be written. The write function takes a single input, the desired output filename. Given the syntax below, the sequence information stored in seqObect will be used to create a .seq file that can then be put into the interpreter module.

```
seqObject.write('mypulseSequence.seq');
```

Using the Interpreter Module

The interpreter module, 'VarianBuild' was designed to be easy to use – it takes just two inputs in string format. The first string is the name of the input file – the file in .seq format that was generated in the step prior. The second string is the desired name of the output file – the one that is a Varian compatible script. An example of the syntax is shown below:

```
VarianBuild('mypulseSequence.seq', 'myVarianSequence.c');
```

This example would create a Varian compatible script named myVarianSequence.c.

Manual Modifications to the Sequence

At times, such as those when an arbitrary waveform is used, the files will require user input – this takes the form as inputting the correctly calibrated RF pulse power levels, or the corresponding Varian shape file name -- the commented code in the generated file will provide a guide for syntax. It should be noted that in future versions, the arbitrary gradient naming process is hoped to be resolved.

Running Scripts on the Scanner

Transferring Sequence to the Scanner

If the user is a student in the ECEN 411 course, the TA will provide instructions on how to transfer the sequence to the scanner. If the user is working at the MRSL, it is in the student's and lab's best interest that they have a more senior student explain the process to them.

Preparing and Running the Sequence

Once the sequence has been placed in the proper folder, psglib, on the scanner, the user should then use the command line and type in 'Ssems' which will navigate to the Ssems macro page. From there, using 'seqgen' in the command line, the user should compile the pulse sequence. The example syntax is shown below:

```
seqgen('myVarianSequence')
```

If the compilation is unsuccessful, it is best to read the errors and review – likely the sequence challenges the scanners limitations. If the compilation is successful, then the user would set their sequence to their sequence using the 'seqfil' command, the syntax for which is shown below:

```
seqfil = 'myVarianSequence'
```

Once set, the file is ready to run. It is recommended that users display their pulse sequence to check it over prior to running, but this is up to the viewer's discretion. At this point, the user can use the on screen 'scan' button on the macro page.

APPENDIX D

Matlab LiveScript Aids

Lab 2 Livescript: Introduction to MR Imaging

```
close all, clear all, clc
```

Pulse Sequence Writing

Lab 2

In this lab, the students run a series of gradient echo sequences with varying TE and TR times.

First, one starts with System Definitions:

```
system = targetMagnet('40cm')
```

```
system = struct with fields:
           maxGrad: 1277280
           maxSlew: []
          riseTime: 5.000000000000000e-04
       rfDeadTime: 1.000000000000000e-04
    rfRingdownTime: 3.000000000000000e-05
       adcDeadTime: 0
      rfRasterTime: 1.000000000000000e-06
    gradRasterTime: 1.000000000000000e-05
```

```
seqObj = mr.Sequence(system)
```

```
seqObj =
  Sequence with properties:

      version_major: 1
      version_minor: 1
    version_revision: 0
        rfRasterTime: 1.000000000000000e-06
      gradRasterTime: 1.000000000000000e-05
         definitions: [0×1 containers.Map]
         blockEvents: {}
           rfLibrary: [1×1 mr.EventLibrary]
         gradLibrary: [1×1 mr.EventLibrary]
          adcLibrary: [1×1 mr.EventLibrary]
        delayLibrary: [1×1 mr.EventLibrary]
        shapeLibrary: [1×1 mr.EventLibrary]
```

If the user wants to use values different than the defaults defined in the targetSystem function, that line can be replaced with an "mr.opts(...)" command, but for the purposes of the ECEN 411 lab, the targetSystem function is used to simplify.

Next, one needs to define some sequence parameters:

```
FOV = 120E-3; % In Meters
Nphase = 128;
Nfreq = 128;
sliceThickness = 3E-3; % In Meters
tramp = system.riseTime; % In Seconds
flip_deg = 20;
```

1

```
flip = flip_deg*(pi/180); % In Radians
TE = 30E-3; % In Seconds
TR = 250-3; % In Seconds
tped = 2E-3; % In Seconds
sw = 20E3;
tacq = Nphase * (1/sw);
gamma = 42.57e6; % MHz/T
phi = 0;
psi = 0;
theta = 0;

seqObj.setDefinition('FieldStrength', 4.7);
seqObj.setDefinition('Nphase', Nphase);
seqObj.setDefinition('Nfreq', Nfreq);
seqObj.setDefinition('tramp', tramp);
seqObj.setDefinition('gamma', gamma);
seqObj.setDefinition('tped', tped);

% Define Euler Angles for Orientation:
seqObj.setDefinition('phi', phi);
seqObj.setDefinition('psi', psi);
seqObj.setDefinition('theta', theta);
```

Next, one can begin the sequence:

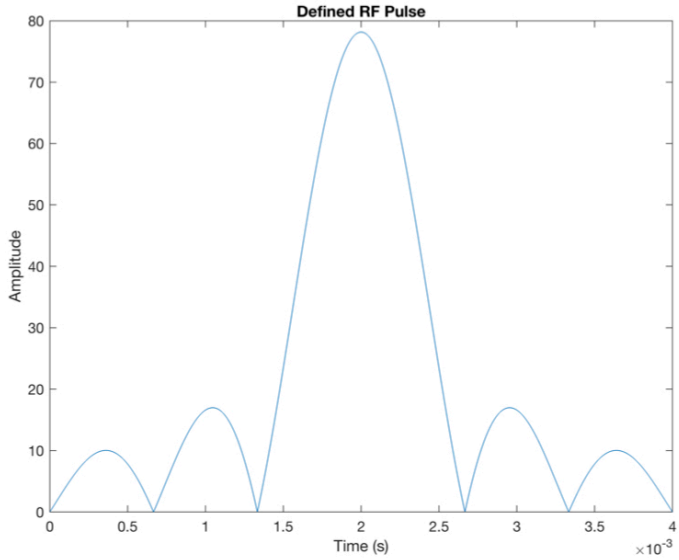
one can begin by defining the RF pulse and the simultaneous slice select pulse. For the purposes of the VarianBuild function, one assign that:

- GX = Readout Direction
- GY = Phase Encode Direction
- GZ = Slice Select Direction

```
rf = mr.makeSincPulse(flip, 'Duration', 4e-3, 'timeBwProduct', 6);
```

one can now see the defined RF Pulse:

```
figure(1)
plot(rf.t, abs(rf.signal));
title('Defined RF Pulse');
xlabel('Time (s)');
ylabel('Amplitude');
```



In order to determine the amplitude of the slice select gradient, one need to determine the bandwidth of the RF pulse, then use that in the equation below:

$$G_{SS} = \frac{\Delta f}{\gamma \Delta z}$$

Note that the gradient amplitudes need to be in terms of Hz/m

```
gss = (6/4e-3)/(sliceThickness); % Hz/m
gssGcm = 100 * gss/gamma
```

```
gssGcm =
   1.174536058256989
```

```
gz = mr.makeTrapezoid('z', system,'flatTime',4e-3,'flatArea',(gss*4e-3));
```

Defining the Readout Gradient

From there, one can calculate the rephase: as equal area, opposite signs with second half of the slice select pulse:

```
gssRephaseArea = -(gz.area)/2
```

```
gssRephaseArea =
    -1.125000000000000e+03
```

```
gssRephase = mr.makeTrapezoid('z', system, 'duration', tped, 'area', gssRephaseArea);
gssRephaseGcm = gssRephase.amplitude * 100/gamma
```

```
gssRephaseGcm =
  -1.761804087385483
```

Next, one move to determine gradient strength in the readout direction. Readout gradient duration is defined by:

$$G_{RO} = \frac{BW}{FOV}$$

Where BW can be defined as the inverse of the acquistion time, Tacq, which one have defined above.

```
gro = 1/(tacq * FOV)  % Hz/m
```

```
gro =
     1.302083333333333e+03
```

```
groGcm = 100 * gro/gamma
```

```
groGcm =
   0.003058687651711
```

```
gx = mr.makeTrapezoid('x', system, 'flatTime', tacq, 'flatArea', gro*tacq);
gxPre = mr.makeTrapezoid('x', system, 'flatArea', -gx.flatArea/2, 'flatTime', tped);
```

Determining Phase Encoding Parameters

Now that the readout gradient and it's dephase has been defined, one can define the gradient amplitude for each phase encoding step using the equation:

$$G_{PE} = \frac{1}{FOV * T_{acq}}$$

```
gpeStep = 1/(tped * FOV);
gpeStepGcm = 100 * gpeStep./gamma;
gpeAmps = ((0:Nphase-1) - Nphase/2) * gpeStep;
```

Creating an ADC Event

Using the array of values, gpeAmps, one can easily call values within a loop to build the sequence. Next one follows by creating the ADC event.

```
adc = mr.makeAdc(Nfreq, 'Duration', tacq);
```

** Insert Gradient Echo Pulse Sequence Diagram Here**

```
delay1 = TE - mr.calcDuration(rf)/2 - mr.calcDuration(adc)/2 - mr.calcDuration(gxPre);
```

```
delay2 = TR - (mr.calcDuration(rf) + TE + mr.calcDuration(gx)/2);
d1 = mr.makeDelay(delay1);
d2 = mr.makeDelay(delay2);
```

Building the Sequence

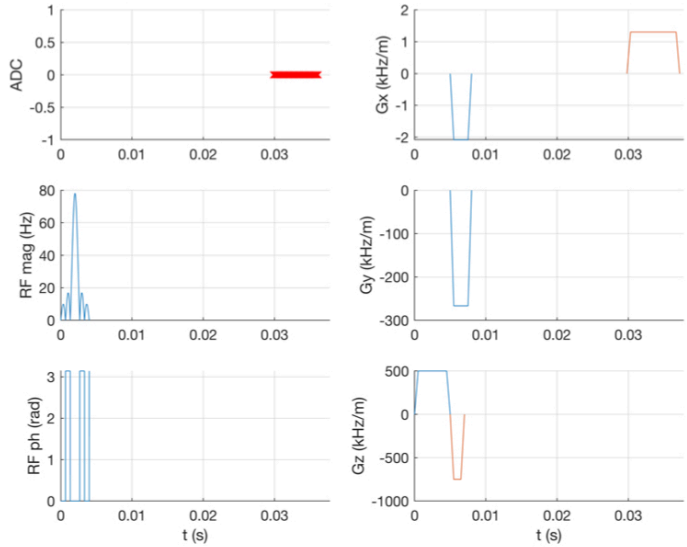
Now, in order to build the sequence with the varying amplitudes in the phase encoding direction

```
for i = 1:Nphase
    seqObj.addBlock(rf,gz);
    gy = mr.makeTrapezoid('y', system, 'flatTime', tped, 'flatArea', gpeAmps(i) * tped);
    seqObj.addBlock(gssRephase, gxPre, gy);
    seqObj.addBlock(d1);
    seqObj.addBlock(gx, adc)
    seqObj.addBlock(d2);
end
```

Sequence Display

Now to display the sequence. The limitation of timebound allows for a single step to be analyzed.

```
format long
timebound = TR-delay2;
timebound = timebound*1.001; % add .1% to time boundary to allow for final value of pulse seque
figure
step = 0;
start = TR*step;
stop = start+timebound;
seqObj.plot('TimeRange', [0 timebound],'type', 'Gradient')
```




```
%suptitle('Gradient Echo Sequence') % suptitle is only available in the bioinformatics toolbox
```

Output File Generation

Finally, in order to create a Varian compatible sequence, first a Pulseq '.seq' must be generated, then can be follooned with a 'VarianBuild' function command. The first parameter is the name of the .seq file, and the second is the desired name of the output file.

```
seqObj.write('Lab2.seq');
VarianBuild_sameDurationGradMod('Lab2.seq', 'temp_2.c');
open('temp_2.c')
```

Pulse Sequence Writing

Lab 3

In this lab, the students observe how modified gradient values interact with the pulse sequence and the image obtained. This lab optionally allows for the students to use a gradient echo or a spin echo sequence. For the ease of the TA, this code will focuse around the spin echo, as it does not require the TA to shim the magnet between lab throughout the day.

NOTE: THIS PAGE HAS MULTIPLE SECTIONS THAT CAN BE RUN AFTER DOING THE INITIAL TWO FOR SET UP. FOLLOWING THAT, THE USER SHOULD SKIP TO THE "WRITE THE SEQUENCE"

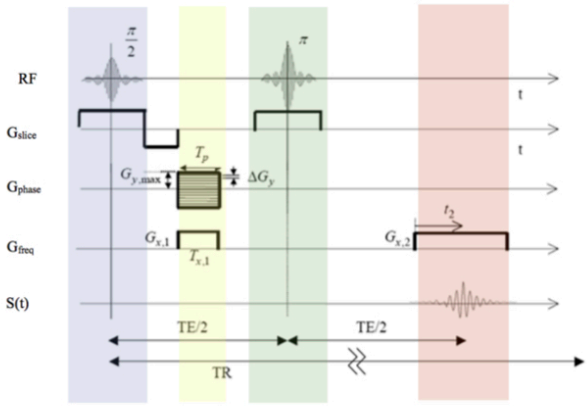
Begin by building the sequence:

Using the same parameters as the previous lab, we can declare our sequence object, as well as imaging parameters.

```
close all, clear all, clc
% Sequence Object Declaration:
system = targetMagnet('33cm');
seqObj = mr.Sequence(system);

% Imaging Parameters
FOV = 120E-3; % In Meters
Nphase = 128;
Nfreq = 128;
sliceThickness = 3E-3; % In Meters
tramp = system.riseTime; % In Seconds
flip = pi/2; % In Radians
TE = 30E-3; % In Seconds
TR = 500E-3; % In Seconds
tped = 2E-3; % In Seconds
tacq = 6.4E-3;
gamma = 42.57e6;
sw=20E3;
phi = 0;
psi = 0;
theta = 0;

% Attaching Parameters to the Sequence
seqObj.setDefinition('FieldStrength', 4.7);
seqObj.setDefinition('FOV', FOV);
seqObj.setDefinition('Nphase', Nphase);
seqObj.setDefinition('Nfreq', Nfreq);
seqObj.setDefinition('SliceThickness', sliceThickness);
seqObj.setDefinition('rampTime', tramp);
seqObj.setDefinition('flipAngle', flip);
seqObj.setDefinition('echoTime', TE);
seqObj.setDefinition('riseTime', TR);
seqObj.setDefinition('gamma', gamma);
seqObj.setDefinition('phaseEncodeDur', tped);
seqObj.setDefinition('tacq', tacq);

% Define Euler Angles for Orientation:
seqObj.setDefinition('phi', phi);
seqObj.setDefinition('psi', psi);
seqObj.setDefinition('theta', theta);
```

Spin Echo Sequence

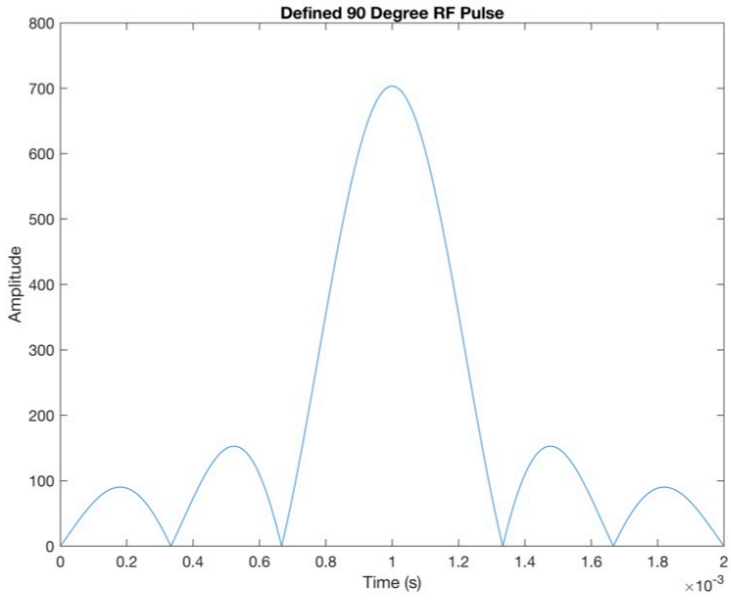
The spin echo Sequence is reproduced below:



Calculation of gradient parameters is the highlight of the lab, but we need to determine the other events to allow for proper definition:

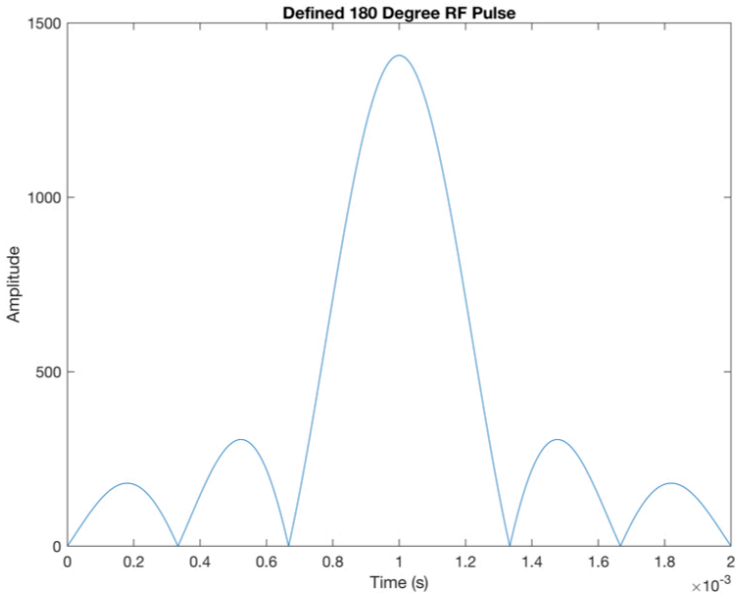
RF Pulse Definition:

```
rf90 = mr.makeSincPulse(flip, 'Duration', 2e-3, 'timeBwProduct', 6);
rf180 = mr.makeSincPulse(flip*2, 'Duration', 2e-3, 'timeBwProduct', 6);

figure(1)
plot(rf90.t, abs(rf90.signal));
title('Defined 90 Degree RF Pulse');
xlabel('Time (s)');
ylabel('Amplitude');
```




```
figure(2)
plot(rf180.t, abs(rf180.signal));
title('Defined 180 Degree RF Pulse');
xlabel('Time (s)');
ylabel('Amplitude');
```

3



Slice Select Gradient Definition:

The students will need to recognize the changing dynamics by comparing the values of G/cm for different

```
gss90 = (6/max(rf90.t))/(sliceThickness);
gss180 = (6/max(rf180.t))/(sliceThickness);

gss90Gcm = 100 * gss90/gamma;
gss180Gcm = 100 * gss180/gamma;

gz90 = mr.makeTrapezoid('z', system,'flatTime',2e-3,'flatArea',(gss90*2e-3));
gz180 = mr.makeTrapezoid('z', system,'flatTime',2e-3,'flatArea',(gss180*2e-3));

gssr = -gz90.area/2;
gzr = mr.makeTrapezoid('z', system, 'flatTime', tped, 'flatArea', gssr);
```

Readout Gradient Calculation

```
gro = 1/(tacq*FOV);
groGcm = 100 * gro/gamma;

gx = mr.makeTrapezoid('x', system, 'flatTime', tacq, 'flatArea', gro*tacq);
gxPre = mr.makeTrapezoid('x', system, 'flatArea', -gx.flatArea/2, 'flatTime', tped);
```

Phase Encoding Gradient Calculation

```
gpeStep = 1/(tped * FOV);
gpeStepGcm = 100 * gpeStep/gamma
```

```
gpeStepGcm =
   0.009787800485475
```

```
gpeAmps = ((0:Nphase-1) - Nphase/2) * gpeStep;
```

Lab Modifications:

Part 1:

Question 2: We want to be able to observe the modifications of the slice select rewinder value:

Demonstrate the purpose of the "rewinder" gradient pulse following a shaped 90 degree pulse? Vary the amplitude or duration of the slice rewind pulse and find the optimal value. Record your data so that you can discuss this step in your lab writeup.

At 90% of the typical value:

```
gssr = 0.90 * gssr;
gzr = mr.makeTrapezoid('z', system, 'Duration', tped, 'area', gssr);
```

At 110% of the typical value:

```
gssr = 1.10 * gssr;
gzr = mr.makeTrapezoid('z', system, 'Duration', tped, 'area', gssr);
```

At 100% of the typical value

```
gssr = 1.00 * gssr;
gzr = mr.makeTrapezoid('z', system, 'Duration', tped, 'area', gssr);
```

Question 3: How to change the slice thickness

What methods could you employ to change the slice thickness? Demonstrate both (the TA will help you change the parameters you want to change). Label the images filename_3a, 3b, etc.

Changing the slice thickness:

```
sliceThickness = 3E-3;
seqObj.setDefinition('SliceThickness', sliceThickness);

gss90 = (6/max(rf90.t))/(sliceThickness);
gss180 = (6/max(rf180.t))/(sliceThickness);

gss90Gcm = 100 * gss90/gamma;
gss180Gcm = 100 * gss180/gamma;

gz90 = mr.makeTrapezoid('z', system,'flatTime',4e-3,'flatArea',(gss90*4e-3));
```

```
gz180 = mr.makeTrapezoid('z', system,'flatTime',4e-3,'flatArea',(gss180*4e-3));

gssr = -gz90.area/2;
gzr = mr.makeTrapezoid('z', system, 'Duration', tped, 'area', gssr);
```

Changing the RF Pulse Duration:

```
rf90 = mr.makeSincPulse(flip, 'Duration', 2e-3, 'timeBwProduct', 6);
rf180 = mr.makeSincPulse(flip*2, 'Duration', 2e-3, 'timeBwProduct', 6);

figure(3)
plot(rf90.t, abs(rf90.signal));
title('Defined 90 Degree RF Pulse');
xlabel('Time (s)');
ylabel('Amplitude');

figure(4)
plot(rf180.t, abs(rf180.signal));
title('Defined 180 Degree RF Pulse');
xlabel('Time (s)');
ylabel('Amplitude');

gss90 = (6/max(rf90.t))/(sliceThickness);
gss180 = (6/max(rf180.t))/(sliceThickness);

gss90Gcm = 100 * gss90/gamma;
gss180Gcm = 100 * gss180/gamma;

gz90 = mr.makeTrapezoid('z', system,'flatTime',4e-3,'flatArea',(gss90*4e-3));
gz180 = mr.makeTrapezoid('z', system,'flatTime',4e-3,'flatArea',(gss180*4e-3));

gssr = -gz90.area/2;
gzr = mr.makeTrapezoid('z', system, 'Duration', tped, 'area', gssr);
```

Question 4: How does the different pulse change the sequence?

```
rfgauss90 = mr.makeGaussPulse(flip, 'Duration', 4e-3, 'timeBwProduct', 6);
rfgauss180 = mr.makeGaussPulse(flip*2, 'Duration', 4e-3, 'timeBwProduct', 6);

figure(5)
plot(rfgauss90.t, abs(rfgauss90.signal));
title('Defined 90 Degree RF Pulse');
xlabel('Time (s)');
ylabel('Amplitude');

figure(6)
plot(rfgauss180.t, abs(rfgauss180.signal));
title('Defined 180 Degree RF Pulse');
xlabel('Time (s)');
ylabel('Amplitude');
```

Part 2:

Question 5: Changing the readout gradient

The TA will assign a field-of-view and bandwidth and enter it in the scanner. You will have to find the readout gradient (frequency encode gradient) and set it on the scanner. Units are in G/cm. Obtain the image by typing “go” in the command window. Using the scan button will change the gradient settings. Save the image (filename_5)

```
BW = 1/(4E-3);
FOV = 8E-3;

gro = BW/FOV;
groGcm = 100 * gro/gamma;

gx = mr.makeTrapezoid('x', system, 'flatTime', tacq, 'flatArea', gro*tacq);
gxPre = mr.makeTrapezoid('x', system, 'flatArea', -gx.flatArea/2, 'flatTime', tped);
```

Question 5: Changing the acquisition bandwidth

The TA will select a different bandwidth (same field-of-view). Repeat the above and save the image. (filename_6)

```
BW = 1/(2E-3);
FOV = 8E-3;

gro = BW/FOV;
groGcm = 100 * gro/gamma;

gx = mr.makeTrapezoid('x', system, 'flatTime', tacq, 'flatArea', gro*tacq);
gxPre = mr.makeTrapezoid('x', system, 'flatArea', -gx.flatArea/2, 'flatTime', tped);
```

Write the Sequence:

First we start with the ADC event:

```
adc = mr.makeAdc(Nfreq, 'Duration', tacq);
```

We need to start by calculating delays:

```
delay1 = TE/2 - (mr.calcDuration(gz90)/2 + mr.calcDuration(gz180)/2 + mr.calcDuration(gxPre));
delay2 = TE/2 - (mr.calcDuration(gz180)/2 + mr.calcDuration(gx)/2);
delay3 = TR - (TE + mr.calcDuration(gx)/2 + mr.calcDuration(gz90)/2);
d1 = mr.makeDelay(delay1);
d2 = mr.makeDelay(delay2);
d3 = mr.makeDelay(delay3);
```

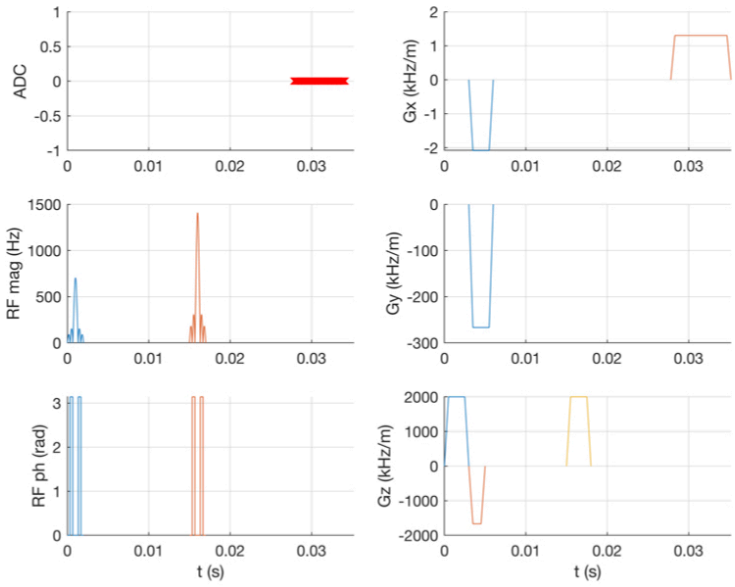
Then we can construct the sequence:

```
for i = 1:Nphase
    seqObj.addBlock(rf90,gz90);
    gy = mr.makeTrapezoid('y', system, 'flatTime', tped, 'flatArea', gpeAmps(i) * tped);
    seqObj.addBlock(gzr, gxPre, gy);
    seqObj.addBlock(d1);
    seqObj.addBlock(rf180, gz180);
    seqObj.addBlock(d2);
    seqObj.addBlock(gx, adc);
    seqObj.addBlock(d3);
```

```
end
```

Now to display and check:

```
format long
timebound = TR - delay3;
timebound = timebound;  % add .1% to time boundary to allow for final value of pulse sequence to be seer
figure
seqObj.plot('TimeRange', [0 timebound],'type', 'Gradient')
```




```
%suptitle('Spin Echo Sequence')
```

Write Sequence to File:

```
seqObj.write('Lab3.seq');
VarianBuild_sameDurationGradMod('Lab3.seq', 'spinEcho.c')
open('spinEcho.c')
```

APPENDIX E

Example Varian Script

Ssems.c – Collected from Varian Scanner System

```
#ifndef LINT
static char SCCSid[] = "@(#)Ssems.c 13.1 10/10/97 Copyright (c) 1991-1995 Varian Assoc.,Inc. All Rights
Reserved";
#endif
/*
 * Varian Assoc.,Inc. All Rights Reserved.
 * This software contains proprietary and confidential
 * information of Varian Assoc., Inc. and its contributors.
 * Use, disclosure and reproduction is prohibited without
 * prior consent.
 */
/*****************************************************************

   Ssems.C  -  2D, multi-slice, Gradient-echo fast imaging sequence
           Uses shaped gradients

           tped - delay for phase encode, slice refocus, and read
                dephase
           tramp - gradient ramp time

           Set imprep='s'; imprep calculates slice parameters only


*****************************************************************************/

#include <standard.h>
#define ID 1.0e-7              /* minimum instruction delay */
static int ph1[4] = {0,2,1,3},           /* 90 deg pulse phase */
              ph2[4] = {1,1,2,2},          /* 180 deg pulse phase, ni=even */
              ph3[4] = {3,3,0,0},                    /* 180 deg pulse phase, ni=odd */
        phr[4] = {0,2,1,3};          /* receiver phase */

pulsesequence()
{
   /* Internal variable declarations ********************/
   double  predelay,seqtime;
   double  temin,tA,tB,tau1,tau2;
   double  gpes,gpeb,groint;
   double  tped,tramp,pi2,gamma;
   double  agss,gssrint;
   double  tic,timin;

   initparms_sis();    /* Initialize parameters for spectroscopy imaging sequences - C */
   tped = getval("tped");  /* look up value of the numeric parameter tped, assign it to tped - C */
   tramp = getval("tramp"); /* look up value of the numeric parameter tramp, assign it to tramp - C */
   pi2 = 3.14/2.0;    /* pi/2 */
```

```
   gamma = sfrq*1e6/B0;   /* nucleus gamma value */  /* Transmitter freq of observe nucleus, divided by
B0*/

   /* slice gradient calculations */
   agss = fabs(gss);  /* take the absolute value of gss - C */
   gssrint = agss*(p1/2.0 + rof2 + tramp/2.0); /* p1 is defined as a pulse width, rof2 is the amplifier blanking
delay -- this value is the integral of gss for the purpose of finding the duration of gssr*/
   gssr = (pi2*gssrint/tped);   /* slice refocussing (sine) gradient */

   if (gssr >= gmax) {
           printf("Error: gssr > gmax; increase thk or tped  \n");
           abort(1);
   }

   if (gss >= gmax) {
           printf("Error: gss > gmax; increase thk or tped  \n");
           abort(1);
   }
   /*** Calculate phase encode gradients ****/
   if (lpe <= 0.0) {
           printf("%s: Error: lpe <= 0.  \n",seqfil);
           abort(1);
   }

   gpes = pi2/(gamma*tped*lpe);    /* phase gradient step size G/cm */
   gpeb = gpes*nv/2;           /* phase gradient base value */
   if (gpeb >= gmax ) {
           printf("Error: gpeb(%7.2f) > gmax; increase tped or lpe \n",gpeb);
           abort(1);
   }

   /*** Calculate readout gradients ****/
   if (lro <= 0.0) {
           printf("%s: lro <= 0 \n",seqfil);
           abort(1);
   }

   gro = sw/(gamma*lro);
   groint = gro*((at+tramp)/2.0);
   gror = (pi2*groint/tped);        /* read dephaser */
   if (gro >= gmax ) {
           printf("Error: gro(%7.2f) > gmax; decrease lro \n",gpeb); // should this be gro??
           abort(1);
   }

   /* tau1 and tau2 are events during each half-echo periods */
   tau1 = (p1+p2)/2.0 + rof2 + rof1 + tped + tspoil + 2*tramp;
   tau2 = (p1+at)/2.0 + rof2 + tspoil + 2*tramp + tped;
   temin = tau1 < tau2 ? 2.0*tau2 : 2.0*tau1;  // if tau1 < tau2, then temin = 2*tau2, if tau1 !< tau2, then
temin = 2*tau1
   if (te < temin) {
           printf("Error: te too short.  Minimum te = %f\n",temin);
      abort(1);
```

```
}
tA = te/2.0 - tau1;
tB = te/2.0 - tau2;

/* Relaxation delay ***********************************/
seqtime = te + rof1 + p1/2.0 + at/2.0 + 2.0*tramp;

/* Inversion time delay */
tic = 2.0e-6;   /* initialize */
if (ir[0] == 'y') {
  timin = (p2+p1)/2.0 + rof1 + rof2 + 2.0*tramp;
  tic = ti - timin;
  seqtime += p2 + tic + tcrush + 2.0*tramp;
  if (ti < timin) {
          printf("Error: ti too short.  Minimum ti = %f\n",timin);
     abort(1);
  }
}

if (tr < seqtime) {
  printf("Error: Requested tr too short.  Min tr = %f\n",seqtime);
  abort(1);
}
predelay = (tr - seqtime)/ns;

initval(nv/2.0,v7);
settable(t1,4,ph1);    /* initialize phase tables and variables */
getelem(t1,ct,v1);     /* 90 deg pulse phase */
settable(t2,4,ph2);    /* 180 pulse during even scan */
settable(t3,4,ph3);    /* 180 pulse during odd scan */
settable(t4,4,phr);
getelem(t4,ct,oph);    /* receiver phase */

/* PULSE SEQUENCE ************************************/
status(A);
obspower(tpwr1);
/* begin phase-encode loop ***************************/
peloop(seqcon[2],nv,v5,v6);
  msloop(seqcon[1],ns,v11,v12);  /* multi-slice loop */

    /* Phase cycle: Part II, p2 alternates each phase step */
    /* to move non-phase-encoded residual fid from 180 *****/
    /* out of image center to phase-encoding edges. ********/
    mod2(v6,v4);
    ifzero(v4);
      getelem(t2,ct,v2);       /* v2 = 1122 */
    elsenz(v4);              /*    or           */
      getelem(t3,ct,v2);       /* v2 = 3300 */
    endif(v4);

    /* Relaxation delay ***********************************/
    poffset_list(pss,gss,ns,v12);
    delay(predelay);
```

```
    xgate(ticks);

    /* Inversion pulse, p2,p2pat */
    if (ir[0] == 'y') {
     obspower(tpwr2);
     obl_shapedgradient("","","ramp_hold", tramp, 0,0,gss, 1,WAIT);
     shapedpulse(p2pat,p2,v2,rof1,rof2);   /* 180 deg pulse */
     delay(tcrush);           /* crusher gradient */
     obl_shapedgradient("","","ramp_down", tramp, 0,0,gss, 1,WAIT);
     zero_all_gradients();
     delay(tic);
    }
  // SS 1, 90 degree
    obspower(tpwr1);
    obl_shapedgradient("","","ramp_hold", tramp,  0,0,gss, 1,WAIT); // ramp up and hold
  shapedpulse(p1pat,p1,v1,rof1,rof2);   /* 90 deg pulse */
    obl_shapedgradient("","","ramp_down", tramp,  0,0,gss, 1,WAIT); // ramp down

    /* Read and phase encode pulse ********/
    /* pe_shapedgradient has a bug! */
  pe2_shapedgradient("hsine", tped, gror,-gpeb,0, gpes,0, v6,v6);
    zero_all_gradients();
    delay(tA);
    /* 180 deg pulse */
    obspower(tpwr2);
    obl_shapedgradient("","","ramp_hold", tramp, 0,0,gss, 1,WAIT);
    delay(tspoil+ID);       /* spoiler gradient */
    shapedpulse(p2pat,p2,v2,rof1,rof2);   /* 180 deg pulse */
    delay(tspoil+ID);          /* spoiler gradient */
    obl_shapedgradient("","","ramp_down", tramp, 0,0,gss, 1,WAIT);
    obl_shapedgradient("","","hsine", tped, 0,0,gssr, 1,WAIT);
    zero_all_gradients();
    delay(tB);

    /* Acquire echo ************************************/

    poffset(pro,gro);
    obl_shapedgradient("ramp_hold","","", tramp,  gro,0,0, 1,WAIT);
    sp2on();
    acquire(np,1.0/sw);
    sp2off();
    obl_shapedgradient("ramp_down","","", tramp,  gro,0,0, 1,WAIT);
         pe2_shapedgradient("hsine", tped, 0,0,gspoil, gpes,0, v8,v8);  /* rewinder and crusher */
    zero_all_gradients();

   endmsloop(seqcon[1],v12);
  endpeloop(seqcon[2],v6);
}

/*************************************************************************************
              Modification History
```

****
**********/

APPENDIX F

Example Script from Interpreter

InterpreterSpinEcho.c

Note: This script contains manual modifications to include the spoiler gradient with the slice select gradient coinciding with the 180° RF pulse and the input of the RF pulse power values.

```
#include <standard.h>

#define ID 1.00000e-07
#define gpeStep 0.007807321
#define gpeBase -0.499660916

static int ph1[4] = {0,2,1,3},
                                ph2[4] = {1,1,2,2},
                                ph3[4] = {3,3,0,0},
                                phr[4] = {0,2,1,3};

static double counter = 0.0;

pulsesequence()
{
double  pi2, gamma, gpeAmp, spectralWidth;

double var_tpwr1, var_tpwr2;

initparms_sis();


pi2 = 3.14/2.0;
gamma = 4.26e+07; /* Replacing: gamma = sfrq*1e6/B0;*/


var_tpwr1 = 37.0;
var_tpwr2 = 43.0;
nv = 128;
spectralWidth = 2.0e+04;

theta = 0.0;
phi = 0.0;
psi = 0.0;


initval(128.0/2.0,v7);
settable(t1,4,ph1);

getelem(t1,ct,v1);
```

```
settable(t2,4,ph2);
settable(t3,4,ph3);
settable(t4,4,phr);
getelem(t4,ct,oph);

/* PULSE SEQUENCE ***********************************/
status(A);
obspower(var_tpwr1);
peloop(seqcon[2],nv,v5,v6);
/* Phase Cycling ************************************/
        mod2(v6,v4);
        ifzero(v4);
                getelem(t2,ct,v2);
        elsenz(v4);
                getelem(t3,ct,v2);
        endif(v4);

/* Phase Encode Gradient Amp Calculation **************/

        poffset_list(pss,0.868304,1.0,v12);
/* Block Number 1 */
                obspower(var_tpwr1);
                obl_shapedgradient("","","ramp_hold",0.000500,    0,    0,0.868304,1,WAIT);
                shapedpulse("sinc",2.000000e-03,v1,rof1,rof2);
                obl_shapedgradient("","","ramp_down",0.000500,    0,    0,0.868304,1,WAIT);
                zero_all_gradients();
/* Block Number 2 */
    gpeAmp = gpeBase + (counter * gpeStep);
                obl_shapedgradient( "ramp_hold", "ramp_hold", "",0.000500, 0.538697013 , gpeAmp ,
0,1,WAIT);
                delay(0.002000);
                obl_shapedgradient( "ramp_down", "ramp_down", "",0.000500, 0.538697013 , gpeAmp ,
0,1,WAIT);
    counter = counter + 1.0;
  zero_all_gradients();
/* Block Number 3 */
                delay(0.006000);
/* Block Number 4 */
                obspower(var_tpwr2);
                obl_shapedgradient("","","ramp_hold",0.000500,    0,    0,0.868304,1,WAIT);
    delay(0.003);
                shapedpulse("sinc",2.000000e-03,v2,rof1,rof2);
    delay(0.003);
    obl_shapedgradient("","","ramp_down",0.000500,    0,    0,0.868304,1,WAIT);
                zero_all_gradients();
/* Block Number 5 */
    obl_shapedgradient("","","ramp_hold",0.000500,    0,    0,0.436766009,1,WAIT);
    delay(0.002);
    obl_shapedgradient("","","ramp_down",0.000500,    0,    0,0.436766009,1,WAIT);
/* Block Number 6 */
                delay(0.0038000);
/* Block Number 7 */
                poffset(pro,gro);
```

```
                    obl_shapedgradient("ramp_hold","","",0.000500,0.391128,    0,  0,1, WAIT);
                    sp2on();
      acquire(256.0,1/spectralWidth);
                    sp2off();
                    obl_shapedgradient("ramp_down","","",0.000500,0.391128,    0,  0,1, WAIT);
                    zero_all_gradients();
/* Block Number 8 */
                    delay(0.9653);

endpeloop(seqcon[2],v6);

}
```